GPU-based, Microsecond Latency, Hecto-Channel MIMO Feedback Control of Magnetically Confined Plasmas

Nikolaus Rath

Submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in the Graduate
School of Arts and Sciences

Columbia University

2013

ABSTRACT

GPU-based, Microsecond Latency, Hecto-Channel MIMO Feedback Control of Magnetically Confined Plasmas

Nikolaus Rath

Feedback control has become a crucial tool in the research on magnetic confinement of plasmas for achieving controlled nuclear fusion. This thesis presents a novel plasma feedback control system that, for the first time, employs a Graphics Processing Unit (GPU) for microsecond-latency, real-time control computations. This novel application area for GPU computing is opened up by a new system architecture that is optimized for low-latency computations on less than kilobyte sized data samples as they occur in typical plasma control algorithms. In contrast to traditional GPU computing approaches that target complex, high-throughput computations with massive amounts of data, the architecture presented in this thesis uses the GPU as the primary processing unit rather than as an auxiliary of the CPU, and data is transferred from A-D/D-A converters directly into GPU memory using peer-to-peer PCI Express transfers. The described design has been implemented in a new, GPU-based control system for the HBT-EP device. The system is built from commodity hardware and uses an NVIDIA GeForce GPU and D-TACQ A-D/D-A converters providing a total of 96 input and 64 output channels. The system is able to run with sampling periods down to 4 µs and latencies down to 8 µs. The GPU provides a total processing power of 1.5×10^{12} floating point operations per second.

To illustrate the performance and versatility of both the general architecture and concrete implementation, a new control algorithm has been developed. The algorithm is designed for the control of multiple rotating magnetic perturbations in situations where the plasma equilibrium is not known exactly and features an adaptive system model: instead of requiring the rotation frequencies and growth rates embedded in the system model to be set a priori, the adaptive algorithm derives these parameters from the evolution of the perturbation amplitudes themselves. This results in non-linear control computations with high computational demands, but is handled easily by the GPU-based system. Both digital processing latency and an arbitrary multi-pole response of amplifiers and control coils are fully taken into account for the generation of control signals. To separate sensor signals into perturbed and equilibrium components without knowledge of the equilibrium fields, a new separation

method based on biorthogonal decomposition is introduced and used to derive a filter that performs the separation in real-time.

The control algorithm has been implemented and tested on the new, GPU-based feedback control system of the HBT-EP tokamak. In this instance, the algorithm was set up to control four rotating $n = 1$ perturbations at different poloidal angles. The perturbations were treated as coupled in frequency but independent in amplitude and phase, so that the system effectively controls a helical $n = 1$ perturbation with unknown poloidal spectrum. Depending on the plasma's edge safety factor and rotation frequency, the control system is shown to be able to suppress the amplitude of the dominant 8 kHz mode by up to 60% or amplify the saturated amplitude by a factor of up to two. Intermediate feedback phases combine suppression and amplification with a speed up or slow down of the mode rotation frequency. Increasing feedback gain results in the excitation of an additional, slowly rotating 1.4 kHz mode without further effects on the 8 kHz mode. The feedback performance is found to exceed previous results obtained on HBT-EP with an FPGA- and Kalman-filter-based control system without requiring any tuning of system model parameters.

Experimental results are compared with simulations based on a combination of the Boozer surface current model and the Fitzpatrick-Aydemir model. Within the subset of phenomena that can be represented by the model as well as determined experimentally, qualitative agreement is found.

Contents

Appendices

Acknowledgments

First and foremost, I would like to thank my wife Tanja, who left friends and family to come with me to the United States so that I could pursue this work. It is also her support that gave me the strength to finish it.

I am also very grateful to my parents for their unconditional support and imperturbable belief in me and all my endeavors.

As my advisor, Michael Mauel has contributed to this thesis in innumerable ways. Above all, it was his unfailing enthusiasm that kept me motivated in times of doubt. Allen Boozer was always willing to discuss any questions I had, and his ideas and explanations are the basis of much of my understanding of theoretical plasma physics. Gerald Navratil's insight and overview of the fusion program has provided focus for much of my research.

My time at Columbia would have not been the same without the other students and technicians in the plasma lab. They provided not only technical and scientific expertise, but also made my time at Columbia much more enjoyable. In particular, I am grateful to Jeffrey Levesque and Daisuke Shiraki for sharing all their knowledge, and Nicolas Rivera and Jim Andrello for their help with technical and mechanical issues.

Chapter 1

Introduction

1.1 Plasma Physics

A plasma is a collection of charged particles that are dense enough to exhibit collective behavior. Plasmas can be found in many different situations. They occur naturally in space as well as on earth, and they are artificially produced for a variety of applications [14]. In space, the interior of stars, the solar wind, galactic nebulae, the interplanetary, interstellar and intergalactic medium are all plasmas of vastly different densities. On earth, plasmas can be found in the ionosphere, the polar aurorae and in lightning strikes. Plasmas are artifically produced for the etching of microchips, to generate pictures in plasma televisions, for electric welding and to generate thrust in ion thrusters.

The plasmas studied in this thesis are quasi-neutral, magnetically confined plasmas. A quasi-neutral plasma consists of similar amounts of positively and negatively charged particles that are well-mixed, so that electric forces only arise on very small scales. A magnetically confined plasma is held within fixed boundaries by a magnetic field. An overview of the physics of such plasmas is given in Boozer [8]. A plasma may also be confined by material boundaries, gravity or by its own inertia. Gravitational confinement requires huge masses, and is therefore only found in astrophysical plasmas. A star is a gravitationally confined plasma. Inertial confinement works only on very short timescales, as it relies on the (very low) mass of the individual particles to resist against acceleration out of the confined zone. Confinement by massive boundaries (i.e., the plasma is contained in a chamber made of some material) occurs only in the laboratory and requires constant replenishment, as the contact with the wall continuously cools and neutralizes the plasma.

1.1.1 Vlasov-Maxwell Equations

A plasma can always be treated as a collection of individual, self-interacting particles that follow their own trajectories and feel just fundamental forces. However, many plasmas can also be treated statistically. In a statistical treatment, particles are not considered individually, but in terms of a distribution function that describes the probability of finding a particle with a given velocity at a given point in space. Conceptually, this is the same approximation that justifies treating water as a fluid rather than a collection of interacting H_2O molecules.

If quantum effects can be neglected and the plasma consists of two particle species of opposite charge (e.g. electrons and ions), the distribution functions $f_\mathrm{i}(\vec{x},\vec{p})$ (for ions) and $f_\mathrm{e}(\vec{x},\vec{p})$ (for electrons), obey the Vlasov and Maxwell equations [46]:

$$\frac{\partial f_\alpha}{\partial t} + \vec{v}_\alpha \cdot \nabla f_\alpha = q_\alpha \left(\vec{E} + \vec{v} \times \vec{B}\right) \cdot \frac{\partial f_\alpha}{\partial \vec{p}} \qquad \nabla \times \vec{B} = \mu_0 \vec{j} + \mu_0 \epsilon_0 \frac{\partial \vec{E}}{\partial t} \tag{1.1}$$

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \qquad \nabla \cdot \vec{E} = \epsilon_0^{-1} \sigma \tag{1.2}$$

$$\nabla \cdot \vec{B} = 0 \qquad \sigma = \int \sum_\alpha q_\alpha f_\alpha \, d^3\vec{p} \tag{1.3}$$

$$\vec{j} = \int \sum_\alpha q_\alpha f_\alpha \vec{v} \, d^3\vec{p} \qquad \vec{v}_\alpha = \frac{\vec{p}/m_\alpha}{\sqrt{1 + p^2/m_\alpha^2 c^2}} \tag{1.4}$$

Here $\vec{B}$ and $\vec{E}$ are the magnetic and electric fields, σ is the charge density, μ_0 and ϵ_0 are the permeability and permittivity of space, q_α is the electric charge of the particles, m_α is the mass of the particles, $\vec{j}$ the electric current density, $\vec{x}$ the position and $\vec{p}$ the momentum.

For a typical plasma that consists of billions of individual particles, this statistical description is a considerable simplification. Making predictions based on it is nevertheless still very hard, because the equations cannot be solved analytically for non-trivial cases, and the complexity of the initial conditions makes it difficult to derive general conclusions from numerical simulations.

1.1.2 Magnetohydrodynamics

In order to get a more tractable set of equations, assumptions need to be made. Depending on the choice of assumptions, different sets of equations can be derived that describe plasma phenomena under different time and length scales.

The approximation used in this thesis is magnetohydrodynamics (MHD) [20]. Its equations are [2]:

$$\rho \frac{d\vec{u}}{dt} = \vec{j} \times \vec{B} - \nabla p + \rho \nu \nabla^2 \vec{u} \qquad \frac{d\rho}{dt} = -\nabla \cdot (\rho \vec{u}) \tag{1.5}$$

$$\vec{E} + \vec{u} \times \vec{B} = \eta \vec{j} \qquad \frac{d}{dt}\left(\frac{p}{\rho^\gamma}\right) = 0 \tag{1.6}$$

$$\nabla \times \vec{B} = \mu_0 \vec{j} \qquad \nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \tag{1.7}$$

$$\nabla \cdot \vec{E} = 0 \qquad \nabla \cdot \vec{B} = 0 \tag{1.8}$$

Here ρ is the mass density, p is the pressure, $\vec{u}$ is the mass velocity, η is the resistivity of the plasma, ν the viscosity and γ is the adiabatic index. It is often useful to assume $\nu = \eta = 0$, and the resulting set of equations is called *ideal MHD*.

Mathematically, these equations can be derived by taking moments of the distribution function for each species [2]. The first moment is the mass density, whose evolution depends on the mass velocity. The second moment gives an equation for the mass velocity in terms of the pressure. By taking the third moment, the pressure could be expressed in terms of the heat flux. However, in order to truncate this infinite chain of equations, one instead assumes that particle velocities follow a Maxwell distribution and the pressure satisfies an adiabatic equation of state. At this point one has two sets of *fluid equations*, one for the electrons and one for the ions. By making the further assumptions that the plasma is approximately neutral, and that, compared to the time scale of ion movement, the electrons are always in equilibrium, one arrives at the MHD equations.

The MHD equations have analytic solutions and are thus very well suited to develop an intuition of the plasma behavior when the underlying assumptions are satisfied. In particular, when looking at time-independent equilibrium solutions, the MHD equations simplify to the single MHD equilibrium equation

$$\vec{j} \times \vec{B} = \nabla p \tag{1.9}$$

1.1.3 Magnetic Confinement

Even though Equation 1.9 is not an exact description of a plasma equilibrium state (because the MHD equations only produce an approximation of the actual plasma dynamics), it is a useful starting point and guiding principle for producing magnetically confined plasmas.

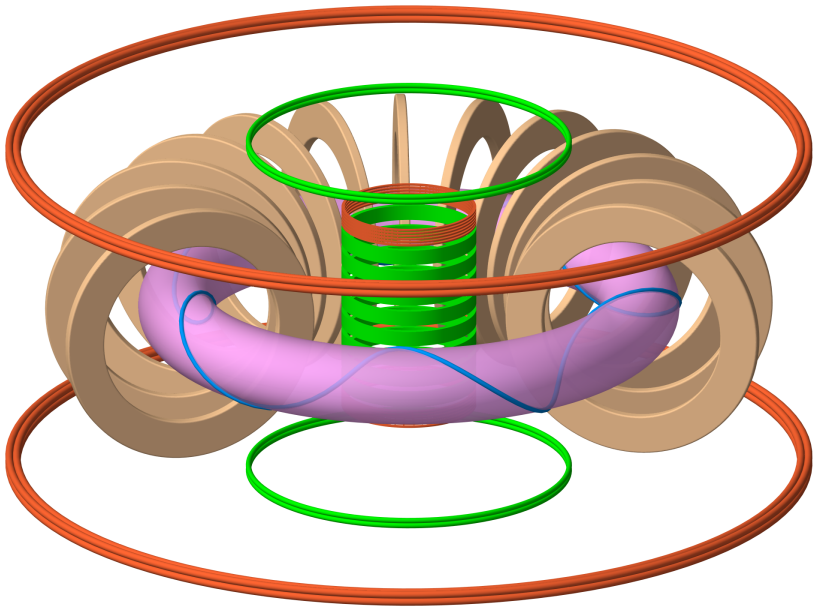

Figure 1.1: Equilibrium coils and magnetic surfaces of the HBT-EP tokamak. Beige: toroidal field coils; orange: vertical field coils; green: ohmic heating coils; blue: single field line wrapping around a toroidal surface (violet). (Figure courtesy of Jeffrey Levesque.)

The idea of magnetic confinement is to use the Lorentz force, $\vec{F} = q\vec{v} \times \vec{B}$, to constrain the motion of charged particles to some closed region of space [26]. The Lorentz force transforms any velocity perpendicular to magnetic field lines into a gyration around the field lines. Magnetic confinement can thus be achieved by either bending the field lines in a such a way that they never leave the desired confinement region, or by establishing some restoring force that constrains the movement parallel to the field lines to some finite length. Neither confinement is perfect, because the necessary completely uniform magnetic fields can not be produced in practice, and the confined particles themselves will change the magnetic and electric fields. Nevertheless, by generating suitable magnetic field configurations, it is possible to create plasmas that satisfy the MHD equilibrium equation 1.9, and are thus, within the approximations of MHD, well confined.

There are multiple ways to achieve magnetic confinement. The approach considered in this thesis is the *tokamak* [60]. In a tokamak, field lines are bent in such a way that they lie on nested, axisymmetric toroidal surfaces. To compensate for the particle drifts due to non-uniform fields, the individual field lines wrap around the surface both toroidally (i.e., the long way around the torus) and poloidally (the short way around). The required fields are

generated by currents in external coils as well as in the plasma. Figure 1.1 shows a schematic picture of the coils and fields in a tokamak.

One of the main driving forces behind magnetic confinement research is the idea to use magnetically confined plasmas for controlled nuclear fusion and harvest the produced energy for electricity generation.

1.2 Nuclear Fusion

Nuclear fusion is the process in which several light nuclei fuse together to produce a heavier nucleus [30, 19]. Nuclear fusion reactions convert an enormous amount of energy (stored as potential energy due to the nuclear binding forces) to heat. For example, the fusion of two hydrogen isotopes to helium releases about 17.6 MeV of energy. For comparison, burning a single molecule of gasoline releases 94 eV, or just about 0.9 eV per atom [19] – less than 10^{-6} of what is freed by a fusion reaction.

Nuclear fusion occurs naturally in stars and is responsible for their heat. On earth, the first bulk nuclear fusion reactions have been achieved in hydrogen bombs. With the declassification of fusion research, an international collaborative effort to harvest the energy produced by fusion reactions for peaceful purposes began. The main challenges in the design of a fusion reactor are maintaining the high temperatures and pressures that are required for fusion reactions to occur, and constructing a vessel that can withstand the constant massive neutron and energy fluxes that are produced [45].

At fusion temperatures and pressures, all elements are in gaseous state and mostly ionized, i.e., they form plasmas. Research on plasma confinement is therefore crucial to the quest of controlled nuclear fusion.

Since gravitational confinement cannot be produced on a laboratory scale, and the heat loss associated with confinement by massive walls is incompatible with the temperatures and pressures required for fusion, fusion plasmas must be confined either magnetically or inertially. Inertial confinement only persists for very small timescales, and the most promising idea of utilizing it is to heat small “target capsules” to fusion temperatures and pressures on a very short timescale with a laser, resulting in a small, controlled explosion. Magnetic confinement can theoretically be sustained for years and thus offers the prospect of a steady-state reactor. The ITER device is a tokamak currently being build in France by an international collaboration with the goal to demonstrate the feasibility of producing fusion

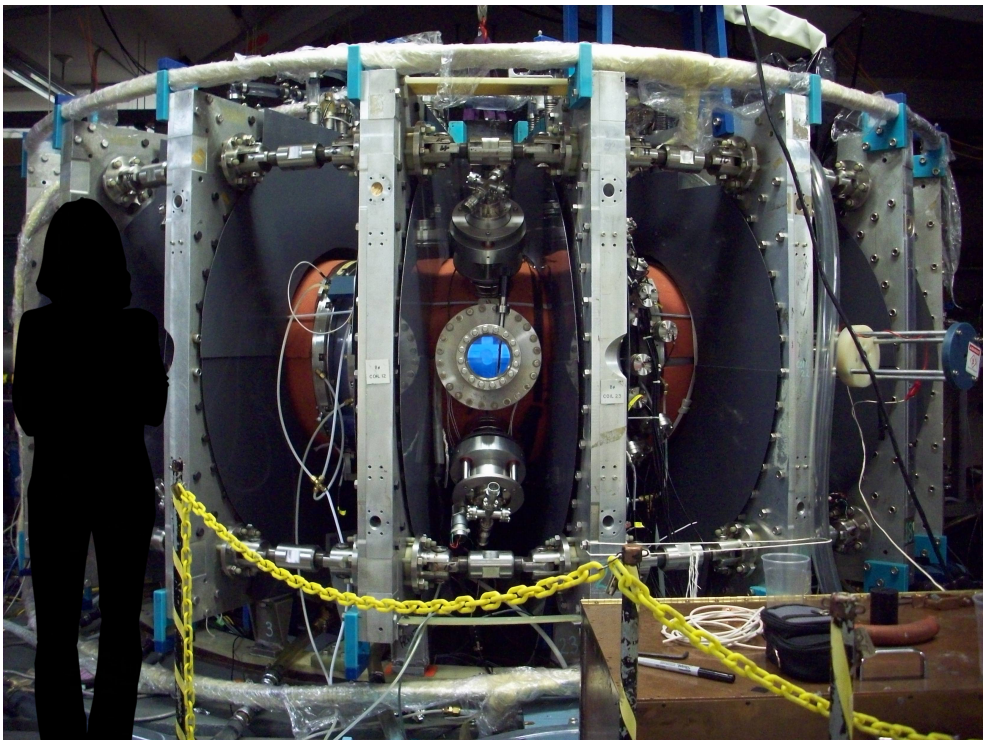

Figure 1.2: The HBT-EP tokamak. The rectangular cases house the toroidal field coils. The white circular coils above and below the case produce vertical fields to stabilize the plasma, which is contained in the vacuum vessel (orange). The blue glow visible through the center window is due to collisions between electrons and neutrals during glow discharge cleaning.

power using magnetic confinement [35].

1.3 HBT-EP

The HBT-EP device [24, 23, 55, 44] is a tokamak located in the Columbia University Plasma Physics Laboratory and has been used in the research for this thesis. In HBT-EP, the toroidal plasma current is induced by rapidly changing the magnetic flux enclosed by the plasma using additional *vertical field (VF)* and *ohmic heating (OH)* coils. Since the plasma has a non-zero resistance, and since coil currents cannot be ramped up indefinitely, HBT-EP plasmas have a limited lifetime of about 20 ms (in larger tokamaks, other techniques compatible with steady-state operation are used to sustain the toroidal current over longer periods). Figure 1.1 shows a CAD drawing of HBT-EP's equilibrium coils and magnetic surfaces. A photograph of

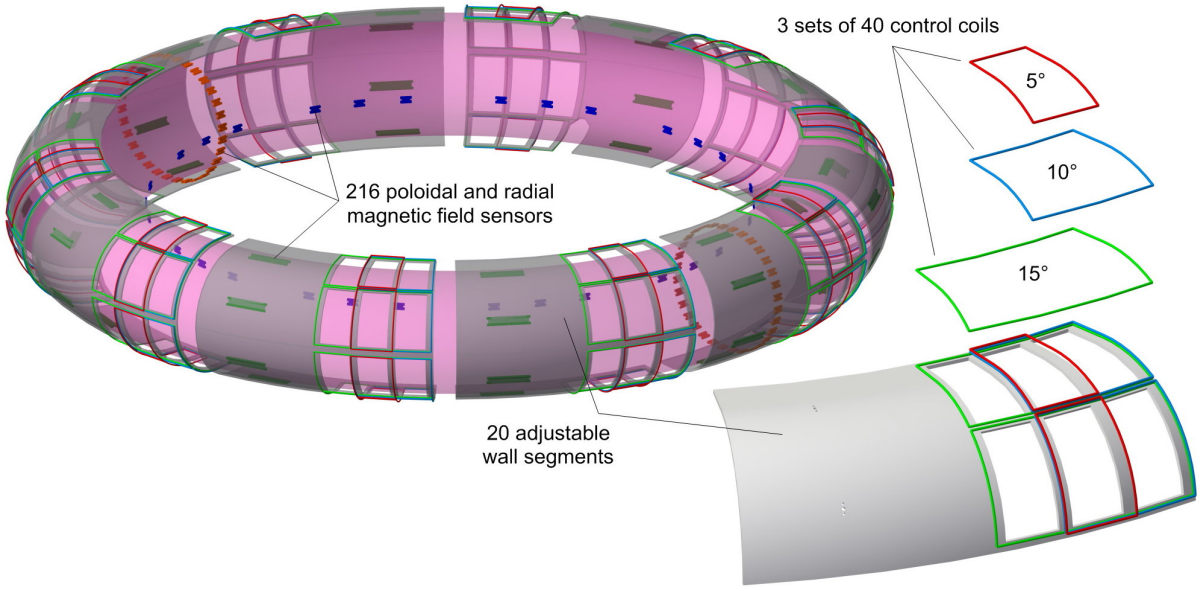

Figure 1.3: Sensors and control coils of the HBT-EP tokamak. Blue "high density" sensors are mounted on the inner wall of the vacuum chamber. Green "feedback" sensors and control coils are located on movable shell segments. Red "high density" sensors are mounted partially on the shell and partially on the wall.

the actual machine is shown in Figure 1.2.

The distinguishing feature of HBT-EP most important for this thesis is a high number of magnetic sensors and control coils, which can be used to detect and adjust the magnetic field configuration in real-time. Figure 1.3 gives an overview over the locations of the different sensors and control coils. A detailed description of the magnetic diagnostics and controls can be found in Shiraki [56] and Levesque [40]. In total, 216 magnetic field sensors are available. With the current signal routing, 80 "feedback" sensors (in green in the figure) can be used for real-time control. Of the 120 control coils, 40 can be used at a time.

1.4 Feedback Control

An MHD equilibrium (given by a solution to Equation 1.9) may be stable or unstable. In the first case, the plasma returns to the equilibrium when slightly perturbed. In the second case, the slightest perturbation will result in an increasingly rapid evolution towards an entirely different, potentially unconfined state. When left to evolve on its own, a plasma will either settle into some stable equilibrium or escape confinement (potentially damaging the

confinement system). In order to ensure that the plasma will settle into not just any, but the desired equilibrium, and to keep it as close to this state as possible even if the desired equilibrium is unstable, the plasma has to be actively controlled from the outside. The general framework in which such kinds of problems are studied is called control theory, and applying control theory to plasma confinement has therefore become an important part of magnetic confinement research.

Generally speaking, control theory is the study of the response of dynamical systems to external inputs [61, 25]. Typically, the goal is to find (and produce) a set of inputs that causes the system to reach a specific reference state. Large branches of control theory are concerned with the control of linear systems. The results can often also be applied to non-linear systems by linearizing them around the reference state. The evolution of many linear (or linearized) systems of interest can be described by a set of ordinary differential equations involving the inputs. In control theory, the canonical way to write these equations is

$$\frac{d\vec{x}}{dt} = \mathbf{A}(t).\vec{x}(t) + \mathbf{B}(t).\vec{u}(t) \tag{1.10}$$

where $\vec{x}$ is the state of the system, $\vec{u}$ the external inputs, and $\mathbf{A}$ and $\mathbf{B}$ are matrices describing the system evolution. Given a reference state $\vec{x}_{\text{ref}}$, control theory deals with finding a set of inputs $\vec{u}(t)$ that will keep $\vec{x}(t)$ as close to $\vec{x}_{\text{ref}}$ as possible.

In control theory, the controlled system is generally referred to as the *plant*. The algorithm (or its physical realization) for computing $\vec{u}$ is called the controller. The physical devices responsible to convert the vector $\vec{u}$ into a physical realization that affects the plant are called *actuators*. When working with magnetically confined plasmas, the plant is typically the plasma itself, and the reference state is an MHD equilibrium described by Equation 1.9.

If $\vec{u}$ is independent of $\vec{x}$, the controller is called an *open loop controller*. If $\vec{u}$ depends on $\vec{x}$ (i.e., the control input depends on the current plant state), the controller is called a *feedback controller* and performs *closed-loop control*.

Typically, the controller does not have knowledge of the plant state $\vec{x}$, but is restricted to some measurements $\vec{y}$ that give limited information about the state. In a linear system, these measurements are determined by two matrices conventionally called $\mathbf{C}$ and $\mathbf{D}$, so that

$$\vec{y} = \mathbf{C}.\vec{x} + \mathbf{D}.\vec{u} \tag{1.11}$$

Physically, the measurements come from *sensors*. The controller has to use the measurements

Figure 1.4: A helical perturbation as it would be produced by a 3/1 resistive wall mode.

$\vec{y}$ to determine the control output $\vec{u}$. How this is done is described by a *control algorithm*. There are many different ways to design control algorithms, and which one works best typically depends on the system that needs to be controlled.

1.5 Resistive Wall Modes

Resistive wall modes (RWMs) are helical perturbations of the plasma from its desired equilibrium state whose dynamics are strongly affected by the distance and resistivity of any conducting structure (the *wall*) that encloses the plasma [17, 5, 22].

The typical RWM is a helical perturbation from a toroidally uniform equilibrium state. Such a RWM is characterized by a unique toroidal mode number n and a spectrum of poloidal mode numbers. Often, one poloidal harmonic m is dominant and the mode is labeled as an m/n mode. The m and n numbers are typically in the single digit range. Figure 1.4 shows an example of a helical perturbation with toroidal mode number 1 and poloidal mode number 3.

Resistive wall modes are stabilized by higher plasma rotation and higher wall conductivity, and driven unstable by high plasma pressures and edge currents. They are least stable when the pitch of the magnetic field lines at the edge of the plasma is close to m/n.

RWMs are an important class of instabilities because they impose a limit on the the maximum achievable ratio of plasma pressure to magnetic pressure (the so-called “beta”). Higher beta, however, is desirable in fusion plasmas for several reasons [35]. Most importantly, it can provide parts of the required toroidal plasma current (via a mechanism called the *bootstrap current*), and corresponds to higher economic efficiency. Suppression of RWMs was therefore one of the early applications of control theory to tokamak plasmas.

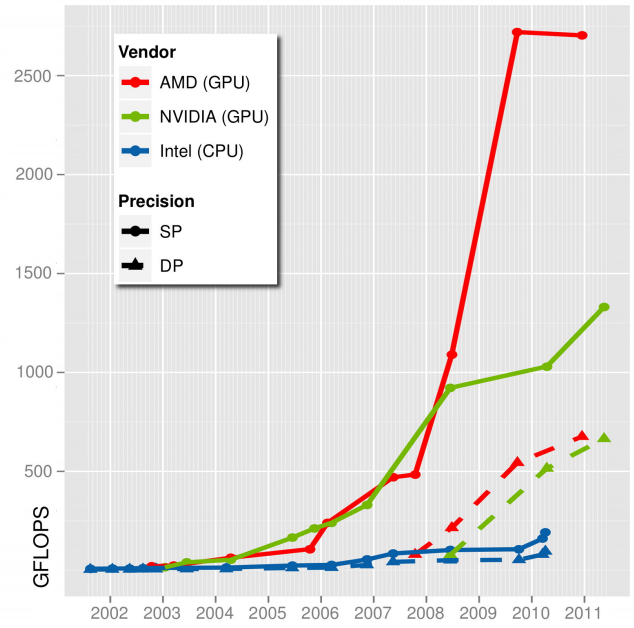

Figure 1.5: Comparison of GPU and CPU performance in terms of GFLOP (theoretical floating point operations per nanosecond) per chip (updated figure from Owens et al. [49]).

1.6 GPU Computing

The development of microchips has followed a trend called Moore's law: the number of transistors on a chip has doubled roughly every two years. This allowed research and development to focus on making individual processings units (*cores*) ever faster and more powerful. In practice this meant that an application could be made to run twice as fast just by waiting for two years and running it on new hardware.

In recent years, this trend has slowed due to increasing problems with heat flow and the transistor size approaching physical limits. In response, development has focused on increasing the number of processing cores available on a chip instead of making individual cores run faster. When the number of cores per chip is factored in, Moore's law is unbroken and expected to hold for the foreseeable future as well.

However, this continued increase in computational power is no longer as easy to harvest for software programmers as before. Unless an application has been written in such a way that it can distribute its operations on multiple processors, it will not run faster no matter how many cores are available.

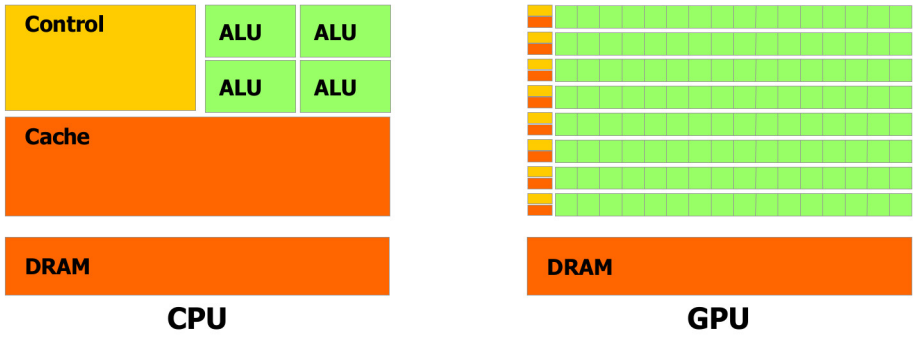

Figure 1.6: Comparison between GPU and CPU architecture. GPUs use most of the available transistors for computing cores (made of *Arithmetic Logic Units* or ALUs) that execute the same operation on different data. CPUs have only few computing cores but sophisticated control logic to speed up the execution of sequential code. Figure from NVidia [47].

Graphics Processing Units (GPUs) are extension cards originally designed to assist the central processing unit (CPU) with the rendering of complex graphics. Since graphics rendering can be easily parallelized, GPUs have come with a high number of relatively simple processing cores for a long time. In recent years, the combined performance of the hundreds of GPU cores has exceeded the total performance of individual CPUs by orders of magnitude (cf. Figure 1.5). This has long been of mostly theoretical interest, because non-graphical applications could not make use of this computational power without major changes. However, with CPU core performance having plateaued, applications now need to be rewritten for increased parallelism also when running on CPUs, so GPUs have become attractive for general purpose computing as well [4, 29, 49]. In response, GPU vendors have added new interfaces and functions that are especially geared for use by the high performance scientific computing.

The large discrepancy between the floating point performance of CPUs and GPUs is due the historically different architecture of GPUs. GPUs implement a computation mode called *SIMT*, for "single instruction, multiple threads" that is optimized for problems with high arithmetic intensity (i.e., the ratio of arithmetic operations to memory operations is very large) and data-parallelism (i.e., the same operations need to be applied to different data). A SIMT processor maintains a large number of threads that all execute the same program (called a *GPU kernel*) but operate on different data. The full theoretical performance can therefore only be achieved if there is enough data that has to be processed the same

way to keep all the available threads busy. The number of threads that can be executed in parallel on a GPU is tens of thousands, but the number of kernels (i.e., distinct programs) that can run in parallel is in the order of tens. A CPU, in contrast, distributes its resources very differently. It implements only a few computing cores, and instead adds extensive data caches and sophisticated flow control. Both of these are intended to speed up the sequential execution of code. Data caches ensure that the few existing cores are kept as busy as possible without having to wait for new data. Flow control allows executing elements of a sequential instruction stream in parallel, while maintaining the appearance of sequential processing. The difference between GPU and CPU architecture is illustrated in Figure 1.6.

GPUs are now widely used in many fields for time- and data-intensive computations where the time required to transfer data to and from the GPU is negligible compared to the time required for the actual computation. This regime begins with real-time computer vision applications like medical imaging [59] and vehicle control [21] that require millisecond response times, and extends up to multi-day simulations of complex physical phenomena like fluid flows [28] and quantum chromodynamics [53]. GPUs have, however, not yet been used for real-time applications in the microsecond regime, where small amounts of data need to be processed extremely fast and the input/output latency becomes an important factor.

However, even in this regime using a GPU is expected to have several advantages over traditional control processing solutions like FPGAs and multi-core CPU computers. A GPU-based system offers more computing power, effectively unlimited input and output channels, and is easier to program and less expensive than FPGAs. Compared to CPU-based systems, a GPU-based system is expected to offer more computing power and better real-time performance without requiring the use of a real-time operating system.

1.7 Overview of the Thesis

This thesis presents a novel control system that, for the first time, uses a GPU for microsecond control computations. The architecture of this system is presented in Chapter 2.

Such a control system can be used for the control of magnetically confined plasmas, and the work presented in this thesis is meant to contribute toward the long-term goal of making such plasmas usable for producing fusion energy.

The proposed architecture has been implemented in a new control system for the HBT-EP tokamak, a magnetic confinement device for plasmas. At HBT-EP, magnetic sensors and

magnetic control coils are used to keep the plasma in a specific 3D shape, and to control and suppress perturbations like RWMs.

The new control system provides actuators, sensors, and computing power, but does not mandate a specific control algorithm. To test the performance of the system at HBT-EP, an *adaptive* control algorithm was developed. This means that the algorithm requires only partial information about the plant, and updates the exact plant model (Equation 1.10) using the measurements. This algorithm is described in Chapter 4, and its implementation on the GPU discussed in Chapter 5.

Chapter 6 and Chapter 7 describe the setup and results of feedback control experiments at HBT-EP with the new control system.

In theory, the effects of any control system and algorithm on the plasma can be simulated. Chapter 8 compares the results of such simulations with experimental observations.

Chapter 2

Control System Architecture[1]

This chapter discusses the integration of digitizers, analog output generators and digital processing units into a novel, GPU-based control system. The main idea presented in this chapter is to use a GPU as the primary computing unit rather than as a subordinate to the CPU. This allows writing of control algorithms with deterministic execution times in mainstream programming languages without the need for a real-time operating system. Further performance increases are achieved by also removing host memory from the control loop and transferring data directly between GPU, digitizers and analog output generators.

After introducing the theoretical design, a concrete implementation for the HBT-EP tokamak is described. This system enables operation with sampling periods down to 4 µs and I/O latencies down to 8 µs, significantly improving over a traditional CPU-based design. Compared to an FPGA controller, the GPU-based system is easier and faster to program while at the same time offering more computing power.

2.1 Overview of Processing Approaches

Control systems currently used in plasma physics are typically based on either FPGA or multi-core CPU systems.

2.1.1 FPGAs

Field Programmable Gate Arrays (FPGAs) are reprogrammable integrated circuits. They consist of millions of interconnected, but independent *logic blocks*, each of which can perform different types of calculations. By re-wiring the connections between the blocks, and changing

[1] Parts of this chapter have been published in Rath et al. [52].

the operations performed by the different blocks, arbitrary programs can be implemented. Once an FPGA has been programmed in this way, it can be used like an application specific integrated circuit implementing the desired logic. In contrast to a microprocessor (which can instantaneously switch between the execution of different programs), changing the program implemented in an FPGA requires an explicit reprogramming that takes several minutes for which the FPGA is unusable. FPGA "programs" thus run on the bare hardware, without any operating system or even the need for the chip to decode instructions. This means that performance is completely deterministic and computations can potentially be implemented highly parallel, with different logic blocks running different operations at the same time. For this reason, FPGA programs are generally not written in mainstream programming languages. Instead, one uses either a very low-level "hardware description language", or very high-level data-flow design tools. The time required for programming and updating an FPGA is therefore very high when compared to a microprocessor. Similarly, the hardware costs for an FPGA chip capable of implementing a non-trivial algorithm is typically several times larger than the cost of a CPU or GPU system.

2.1.2 CPUs

Multi-core CPU systems are familiar to most people as standard desktop and server computers. These systems use a comparatively low number of microprocessors to execute arbitrary code. A microprocessor has no hard-wired functionality, its input consists of both data and instructions on what to do with the data. The instruction sets typically do not allow much parallelism, so that a microprocessor is essentially executing one instruction after the other. This means that a microprocessor has to be very fast: while FPGA clock rates are in the order of MHz, high-end microprocessors are in the low GHz range. The advantage of multi-core CPU systems are that they are very easy to program and re-program. The one-instruction-at-a-time operation makes it easy to break problems into small consecutive steps that are straightforward to implement. Since a microprocessor executes a stream of instructions and data, the instruction stream can even be changed on the fly and by the program itself. Compared with an FPGA, where every change requires a shutdown and slow reprogramming, using one (or more) microprocessors allows quick testing and debugging of the implementation. Most mainstream programming languages are designed for the programming of microprocessors.

The drawback of using a CPU-based system for computations is that performance is

non-deterministic, and that much of the simplicity of programming is lost when programs need to scale to multiple processors: when a program is not running fast enough on a single CPU, and the clock frequency cannot be increased any further, the only way to speed up to execution is to distribute computations among multiple CPUs. This requires major changes to the program code which eliminate much of the ease of programming (that comes from the sequential execution that can be relied upon when using only a single CPU). The unpredictability of execution times arises from the fact that modern CPUs simulate parallel execution by quickly switching between instruction/data streams from different programs. Which program gets executed when and for how long depends on the usage pattern in complicated ways. Executing the same program to solve the same computation may thus take different amounts of time on every execution. For long-running programs, this effect is typically averaged out. However, for computations that take less than about a millisecond, execution times can vary by orders of magnitude. Special "real-time" operation systems are available and can guarantee specific response times, but their use introduces additional complications.

2.1.3 GPUs

The motivation for the design of a GPU-based plasma control system was to develop a system that gives access to the enormous parallel computing power of GPUs, and combines the advantages of FPGA and CPU-based solutions by also offering fully deterministic execution times and being easy enough to program for programming to be done by plasma physics researchers rather than computer science experts.

As mentioned in Section 1.6, GPUs are already widely used in many fields for time- and data-intensive computations. However, these are generally applications where the time required to transfer data to and from the GPU is negligible compared to the time required for the actual computation. GPUs have not yet been used for real-time applications, where small amounts of data need to be processed extremely fast and the input/output latency becomes an important factor. This is also reflected in the basic GPU design: cores are optimized for high instruction throughput rather than fast execution of individual operations, and GPU cores are typically used as subordinate cores with the CPU being in charge of the main computation.

Nevertheless, GPUs are a very promising technology for control systems as well. The huge number of processing cores allows a level of parallelism that easily matches the one

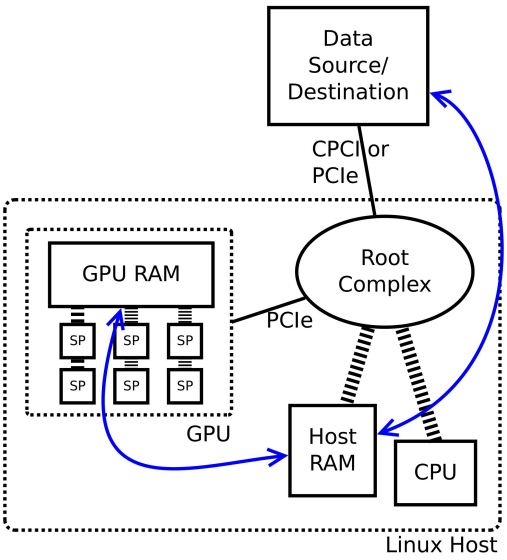

Figure 2.1: Traditional usage of GPUs in scientific computing. The CPU transfers data to GPU memory via a bounce buffer in host RAM, starts the GPU kernels, waits for them to finish, reads the results back into the bounce buffer and then transfers them to the final destination. Blue arrows indicate the logical data flow, while black lines represent the physical connections.

offered by FPGAs. In contrast to FPGAs, however, GPUs can be programmed in established programming languages. They also offer orders of magnitude more computing power and memory for a fraction of the price.

Compared to a CPU-based approach, a control algorithm running on a GPU has a much more deterministic performance without the need for a real-time operating system, because the GPU does not switch between different programs. Furthermore, the higher number of cores allows a much higher level of parallelism and thereby also significantly more computing power.

2.2 Traditional GPU Computing

The classical way to use GPUs in scientific computing is to run the main application on a CPU, and then offload specific computations to the GPU. This setup is illustrated in Figure 2.1. A characteristic feature of this approach is that every action is initiated by the CPU, and all data passes through host RAM to get to its final destination. This setup works very well as long as the time required for computation is significantly longer than the time required for

transferring data, as well as significantly longer than the average CPU scheduling granularity.

In a plasma control system, however, the processing times can be on the order of microseconds, and individual data packets are often less than a kilobyte in size. The latter means that even though the PCIe bus has a typical bandwidth of several GB/s, the time required for transfer of a data packet is dominated by the latency of setting up the transfer, which is in the order of microseconds as well. Finally, on a standard Linux system, it can take hundreds of microseconds for even a high-priority task to be scheduled on the CPU. Taken together, this means that the control system latencies would be dominated by several redundant data transfers and control sampling periods limited by the CPU scheduling granularity.

2.3 GPU-Exclusive Computing

The first step in adapting traditional GPU computing for use in microsecond regimes is the transfer of program flow control from the CPU to the GPU. This can be realized in software by suitable implementation of the control algorithm. Traditionally the CPU defers specific calculations to the GPU, provides data for it and collects the result. In order to avoid latencies due to CPU scheduling, the control algorithm thus needs to be implemented in such a way that the CPU is only involved once when the control system initializes. In this case the CPU sets up memory for use by the GPU, but is not responsible for providing data in the input buffer or collecting data from the output buffer. This computing approach will be referred to as *GPU-exclusive* computing, and is illustrated in Figure 2.2.

By moving the main control loop from the CPU into the GPU, the control system becomes independent of the CPU scheduling performed by the operating system. Since the GPU kernels have exclusive access to the GPU processing cores, their run time is fully deterministic. This means that no real-time operating system is required and the CPU can even be used to perform different tasks (as long as they do not cause excessive load on the PCIe bus).

It should be noted that the GPU-exclusive computing approach is not a general replacement for the traditional, CPU directed approach, but designed for the specific use case of microsecond computing times. For longer, more complex computations, the GPU-exclusive approach will generally impose unacceptable constraints, because both the size and the operations that can be carried out by a GPU kernel are limited. For example, as of 2012 a GPU kernel is not allowed to change the number of concurrent threads, or dynamically manage memory. The kernel thus has to work with the number of threads and the memory resources

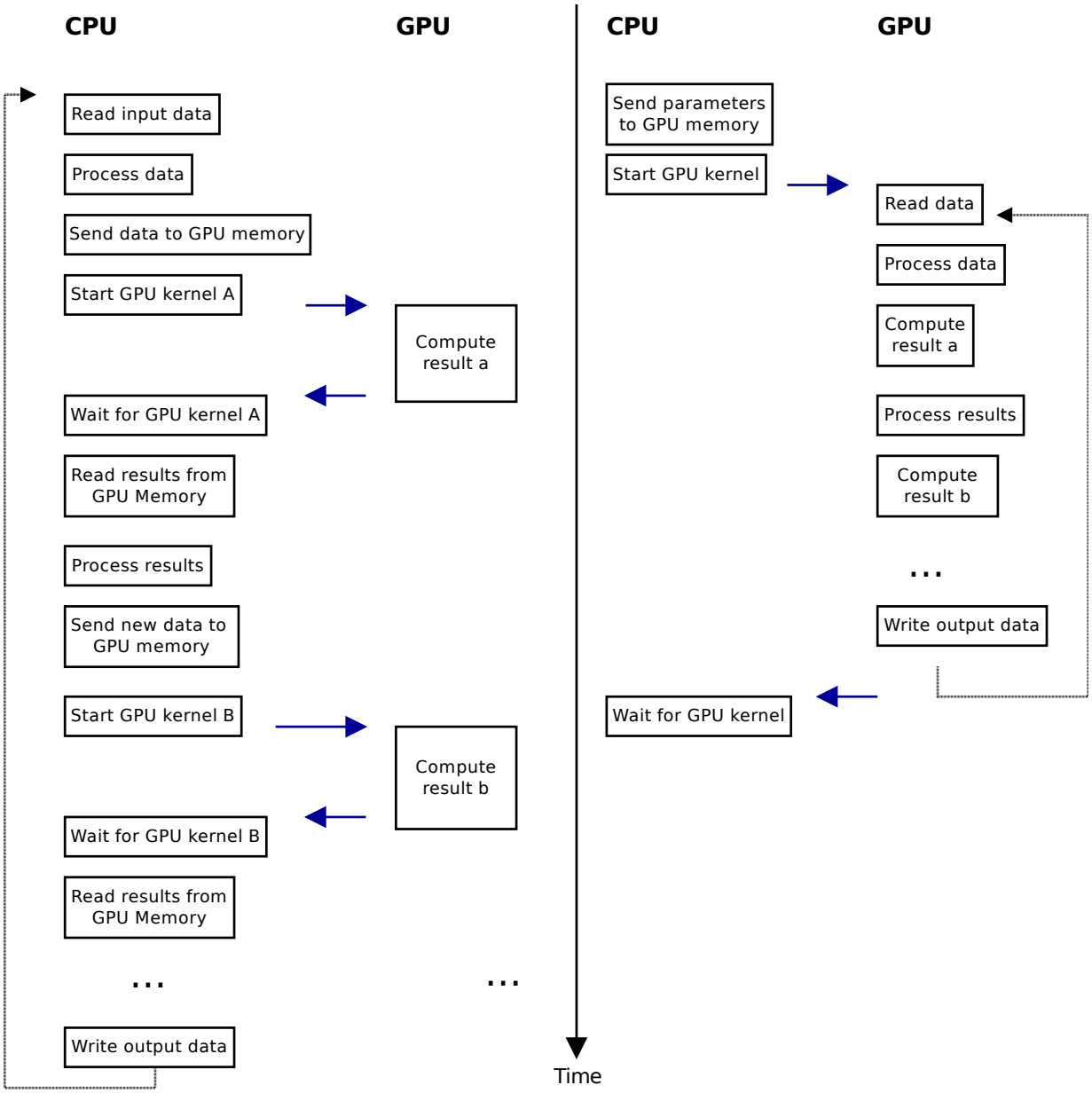

Figure 2.2: Comparison between GPU-exclusive (right) and traditional (left) GPU computing. In the traditional case, the CPU delegates specific calculations to the GPU and collects the results. In the GPU-exclusive case, the CPU is merely used to start the GPU kernel, which is then responsible for all further processing. Blue arrows indicate synchronization steps.

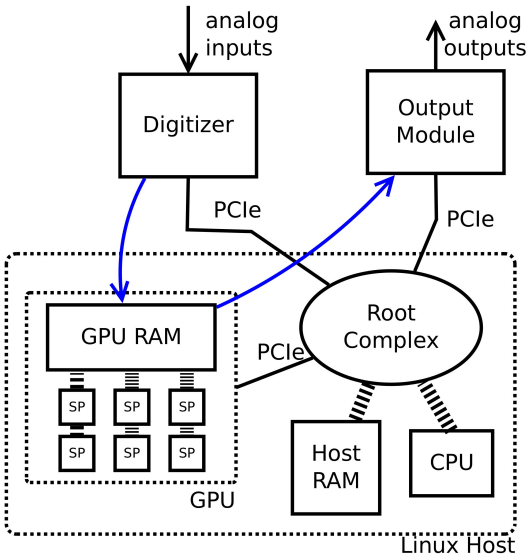

Figure 2.3: Architecture of a GPU-based control system suitable for microsecond regimes. Blue arrows indicate the logical data flow, while black lines represent the physical connections. The digitizer pushes data directly into GPU RAM, and the analog output modules pull data directly from GPU RAM. All components operate independently, and host RAM and CPU are not involved in the control cycle at all.

that it was started with. In order to be implemented entirely in a GPU kernel, an algorithm may thus not exceed some complexity threshold. Luckily, as will also be seen in later chapters, typical control algorithms are well below this limit and are well suited for implementation as GPU kernels.

2.4 Peer-to-Peer DMA Transfers

The transfer of control from CPU to GPU at initialization of the control system allows deterministic performances independent of CPU scheduling. However, in order to be suitable as a control processor, algorithms need not only execute in constant times, but must also be sufficiently fast. As explained earlier, for typical data sizes and algorithms in a control system, a significant fraction of the processing time is spend on the transfer of data between digitizers, memory, processing cores and output modules. The second step in adapting traditional GPU computing for use in control systems is therefore to eliminate the latency introduced by redundant data transfers.

The control system presented in this thesis achieves this by using new, different architec-

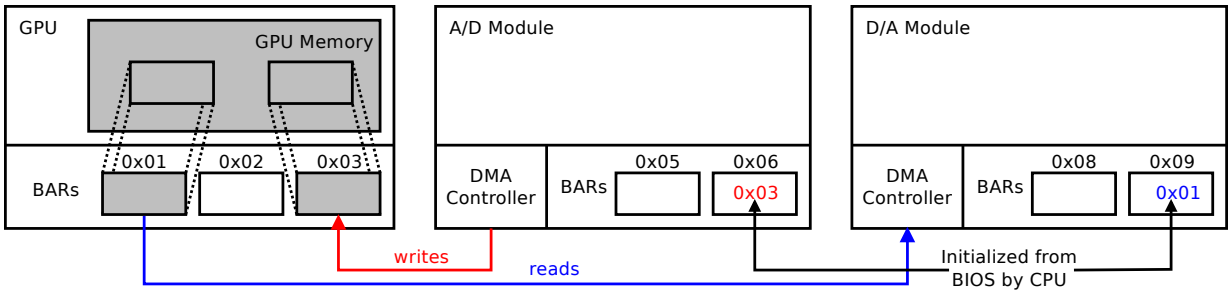

Figure 2.4: Setup procedure for peer-to-peer DMA transfers. BARs are assigned to each PCIe device on system boot. The CPU retrieves these addresses from the BIOS, and communicates the GPU's BAR to the AD/DA modules (by writing the address *into* their BARs). AD/DA modules can then directly address GPU memory mapped into the GPU BAR.

ture which is shown in Figure 2.3. As explained in Section 2.3, the CPU is used to initialize the control system, but then neither host RAM nor CPU are used while the control system is running. Instead of transferring data to a bounce buffer in host RAM and then waiting for the CPU to initiate the transfer from the bounce buffer to GPU memory, the data source (in this case a digitizer) pushes data packets directly into GPU memory via the PCIe bus. Similarly, the control output is pulled directly from GPU memory by analog output modules. Once all components (digitizer, analog output modules and GPU) have been initialized by the CPU, they operate independently and concurrently.

2.4.1 Setup Procedure

The setup procedure for peer-to-peer DMA transfers is illustrated in Figure 2.4. At system boot, the BIOS assigns specific PCI address ranges to the GPU and the AD/DA modules. These *base address ranges* (BARs) are queried by the CPU and used to initialize the devices. First, the GPU needs to map part of its internal memory into one its assigned BARs. With a suitable PCIe root complex (that supports peer-to-peer transfers), this memory can then be directly read and written by any other PCI device that knows the correct address. With the mapping established, the CPU then communicates the addresses of the GPU memory regions for control input and control output to the AD/DA modules (using their BARs for communication). These modules can then read their input and write their output directly from the GPU.

After all initializations have been completed, the CPU starts the GPU cores which do all

further processing. On every clock tick, (1) the digitizer acquires a new sample packet and writes it into GPU memory and (2) the analog output modules read output data from GPU memory and update their outputs. The GPU cores continuously wait for new data to arrive in the input memory region, process the data, and write control output into the output memory region as soon as it is ready. All samples are sequentially numbered, so that GPU and output modules are able to detect if a sample has been missed.

2.5 Implementation for HBT-EP

In the past, the HBT-EP tokamak has used a control system based on National Instruments R-series hardware [34]. Each R-series board offered 8 analog inputs and outputs connected to a LabView programmable FPGA chip. Each board operated independently, so the control system consisted of 4 units that independently tracked state and generated control output based on different sets of 5 input signals each.

Recently, HBT-EP has been equipped with a significantly increased number of magnetic sensors and control coils [44]. In this context it was also decided to switch to an unified controller that can make use of all input signals to estimate the state and generate all outputs, thereby paving the way for detection and control of multiple and more complex perturbations.

Recent LabView versions support programming the FPGA chips for DMA transfers, so that they can indirectly talk to each other over the PCI bus. However, the communication overhead of such a transactions is in the order of microseconds, so that transferring input and output for 40 signals would add about 20 μs of latency. In addition to that, in such a setup, the FPGA chips on all but one board would remain unused, and algorithms would be likely to hit complexity limits due to the limited number of FPGA gates on one chip. For these reasons, replacing the old control system with a GPU-based system was a natural choice, and the architecture described above has been used to build and test a new, GPU-based plasma control system for the HBT-EP tokamak.

2.5.1 Components

The GPU-based control system for HBT-EP was built from the following components:

- NVIDIA GeForce GTX 580 FTW GPU (512 cores, 1.5 GB RAM)
- D-TACQ ACQ196 96 channel, 16 bit digitizer

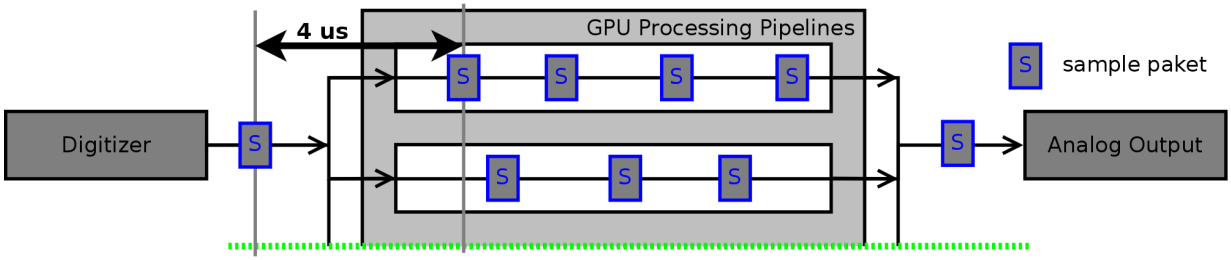

Figure 2.5: Data flow in a GPU-based control system. The sampling period is the spacing between subsequent sample packets. In tests, sampling periods down to 4 µs were achieved.

- Two D-TACQ AO32CPCI 32 channel, 16 bit analog output modules
- Supermicro WhisperStation host system with 3 One Stop Systems PCIe-HIB2-x1 host bus adapters, running a 64bit Linux system with kernel 3.0.0.

The GPU is directly integrated into the WhisperStation, and the three D-TACQ modules are fitted into a 2U CPCI chassis. The CPCI chassis is used only for housing and power supply, and the integrated PCI bus is not used. Instead, the three host bus adapters are connected to the D-TACQ modules with PCIe cables to provide direct connections between all components.

2.6 Performance

The most important measures for the performance of a control system are its latency, sampling period and computing power. This section introduces these metrics and presents measurements obtained from the system implemented for the HBT-EP tokamak.

The sampling period is the rate at which the control system reads new input samples and updates its output signals. The smaller the sampling period, the more smooth the control output. This is illustrated in Figure 2.5. It should be noted that once a data packet has been sent to the GPU, all processing can be pipelined and parallelized, so that the achievable sampling period is effectively independent of the control systems computing power and latency.

The input/output latency of the control system is the time delay between a change in the (analog) control input and the corresponding change in the (analog) control output. The lower the latency of a control system, the faster it can react. Figure 2.6 illustrates how the latency can be measured. In this figure, the control system was set up to forward control

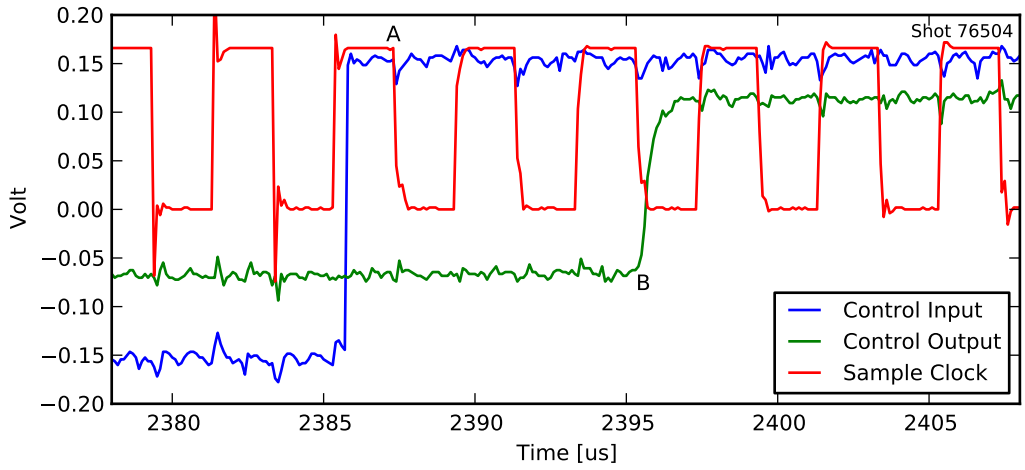

Figure 2.6: The control system latency is the time that passes between a change in the control input (blue) and the corresponding change in the control output (green), taking into account that the input is only sampled on downward edges of the (red) sample clock. The relevant delay in this case is 8 µs and indicated by "A" and "B". (Control input and output do not agree exactly in amplitude and offset due to wire resistances and digitizing offsets.)

input signals unchanged as control output. Control input, control output and the sample clock were digitized on a high-speed digitizer and are plotted in the figure. The (blue) control input is sampled by the control system digitizer on every downward edge of the (red) sample clock. At the same times, the control system's analog output modules pull updated output data from the GPU to update the (green) control output signals. In this case, the control input is a square wave generated with a function generator. The control system latency can then be read off as the interval from the first downward edge of the sample clock after a sign change of the control input to the corresponding sign change in the control output. In Figure 2.6, these points are labeled with "A" and "B", and the latency of the system is 8 µs.

2.6.1 Baseline Latency

For the first performance test, the control system was set up with the same buffer serving as both input and output buffer. The latency was then determined as a function of sampling period, first with the buffer residing in host memory, and then in GPU memory. In this setup, no computation is done at all, every control input as passed through as control output. The measured latencies therefore establish a lower limit on the times achievable by a real control algorithm and indicate the time that is required for analog to digital conversion, transfer of

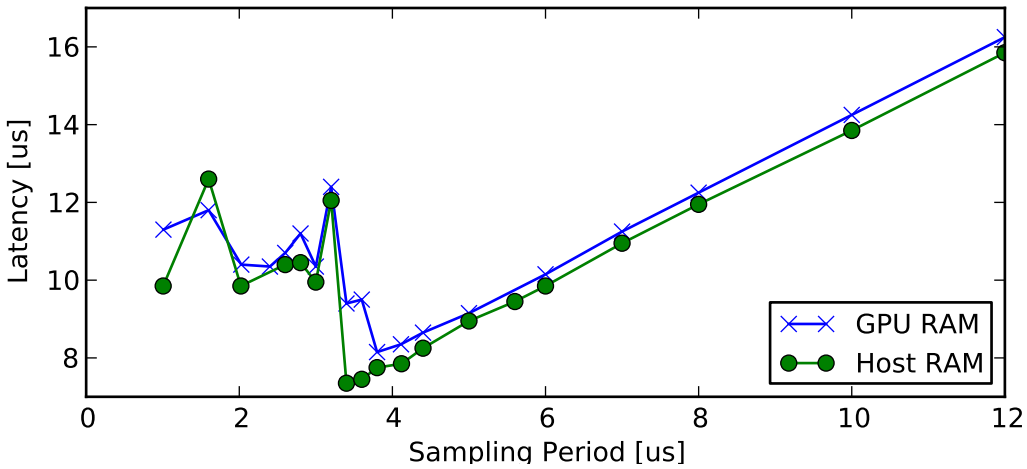

Figure 2.7: Input/Output latency for different sampling periods when transferring through GPU and host memory. The offset of the linear region indicates the time required for transfer to the analog output module and D-A conversion and is 4 µs. The location of the jump indicates the time required for A-D conversion and transfer from the digitizer and is 3.5 µs.

the data from digitizer to a buffer in GPU or host memory, transfer back to the analog output module, and digital to analog conversion.

The results are plotted in Figure 2.7. In this mode of operation, the results for GPU and host memory are very similar. This is expected because there is no significant difference between the two locations as long as no computations are performed. However, several other important pieces of information can be inferred from this plot. Excluding the region below 3.5 µs, the latency is linear in the sampling period. Since input and output data are pushed and pulled at the same time, the offset of this line indicates the time required for pulling data from memory and D-A conversion and is 4 µs. When the sampling period is decreased below 3.5 µs, the latency suddenly jumps up by an amount equal to the sampling period. This indicates that with sample times below 3.5 µs, the D-A modules are pulling samples before they have been received from the digitizer. Therefore, the output suddenly lags behind by a full sampling period. The sampling period at which this jump occurs thus tells us how much time is required for D-A conversion and transfer into the input buffer. In the region to the left of 3.5 µs, the latency appears erratic, because the AD/DA modules are reaching their limits as they would have to convert multiple samples at the same time.

As expected, the minimum latency for the HBT-EP control system is therefore $3.5 + 4 = 7.5$ µs at a sampling period of 3.5 µs. It should be noted that this is a lower bound that does

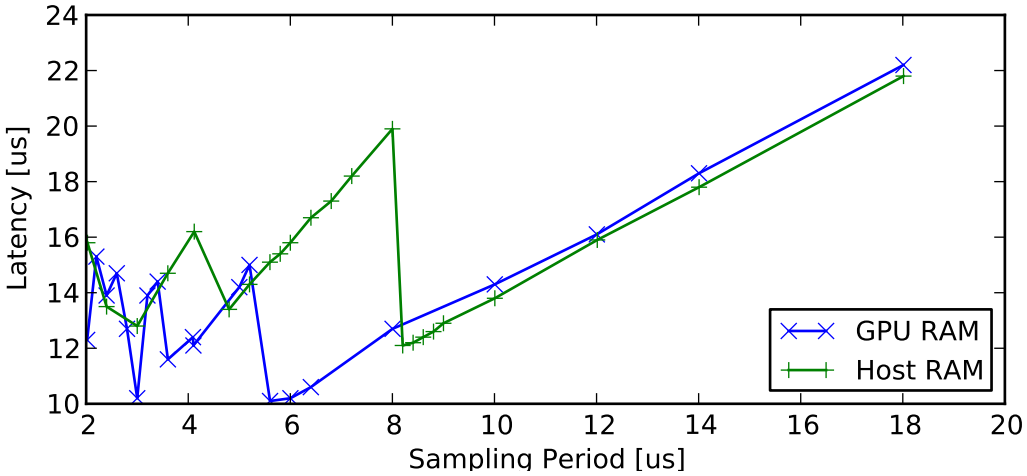

Figure 2.8: Input/Output latency for different sampling periods in active copy operation using host and GPU memory. When using host memory, the time required for digitization and transferring to the input buffer changes from 3.5µs to 8 µs and significantly increases latency. When using peer-to-peer transfers to GPU memory, the transfer time increases less to 5.5 µs, and no additional latency is introduced.

not yet take into account any control computations, so actual latencies will be higher. These effects are quantified in the next sections.

2.6.2 Improvements from peer-to-peer transfer

For the second performance test, the control output was still a copy of the control input. However, now separate buffers were used for input and output samples, and the GPU was used to actively copy input to output. The results of this test therefore provide information about the time required for transfer between input/output buffers and the GPU cores. In an ideal system, this time would be negligible and the latency vs sampling period plot identical to the one in Figure 2.7.

The minimum latency that can be achieved at different sampling periods when using GPU memory and host memory to actively copy input to output is plotted in Figure 2.8. Consider first the green trace, for which input and output buffers are located in host memory. Here the data has to flow through the PCI Express bus four times, as indicated in Figure 2.1: from digitizer to input buffer and from input buffer to GPU, and then from GPU to output buffer, and from output buffer to analog output module. This results in additional latency: it now takes 8 µs instead of 3.5 µs until input data is available in the output buffer. The minimum

latency of such a system is therefore increased from 10 µs to 15 µs at 6 µs sampling period, and from 12 µs to 20 µs at 8 µs sampling period.

Consider now the optimized architecture represented by the blue trace in Figure 2.8, and illustrated in Figure 2.3. Here the data has to pass through the PCIe bus only twice: from digitizer to GPU memory, and from GPU memory to the analog output module. As indicated by the location of the jump in the latency/sampling period plot, the time until a sample arrives in the input buffer is now increased by 2 µs to 5.5 µs rather than by 4.5 µs to 8 µs as in the traditional architecture. This results in significant performance improvements: both at 6 µs and at 8 µs sampling period, the latency is now equal to the lower limit of 10 µs and 12 µs (as defined by the baseline latency)! Compared to the traditional architecture, the use of peer-to-peer DMA transfer has thus reduced the latency from 15 µs to 10 µs at 6 µs sampling period, and from 20 µs to 12 µs at 8 µs sampling period.

2.6.3 Computing Power

In the last two sections, I have established minimum latencies for the control system that are imposed by the time required for AD/DA conversion and transfer to the processing unit. This section examines the last metric of control system performance: the computational power. The total latency is the sum of the minimum latency and the time required to process individual samples. Therefore, the higher the computational power of the system, the lower the total latency.

To assess the performance of the control system hardware, a benchmark control algorithm was implemented for both GPU and CPU. Plans to also benchmark against the FPGA-based system had to be abandoned for reasons that will be described later.

The benchmark algorithm applies an n by n matrix to the control input three times. The result is used as the control output. The matrix is an identity matrix with 1% random perturbations, which ensures that latency can still be measured as before using a square wave input. Even though the number of inputs send by the digitizer is limited to 96, and the number of outputs retrieved by the output modules limited to 64, the algorithm is able to run with n exceeding both numbers. In this case, input and output buffers are allocated with the required size, but not completely read and written by the AD/DA modules.

Every sample that is send by the digitizer into the input buffer includes a sequential sample number. If the control processor is not able to finish computation of a sample before the next one arrives (i.e., in the sample period), this can therefore be detected by the control

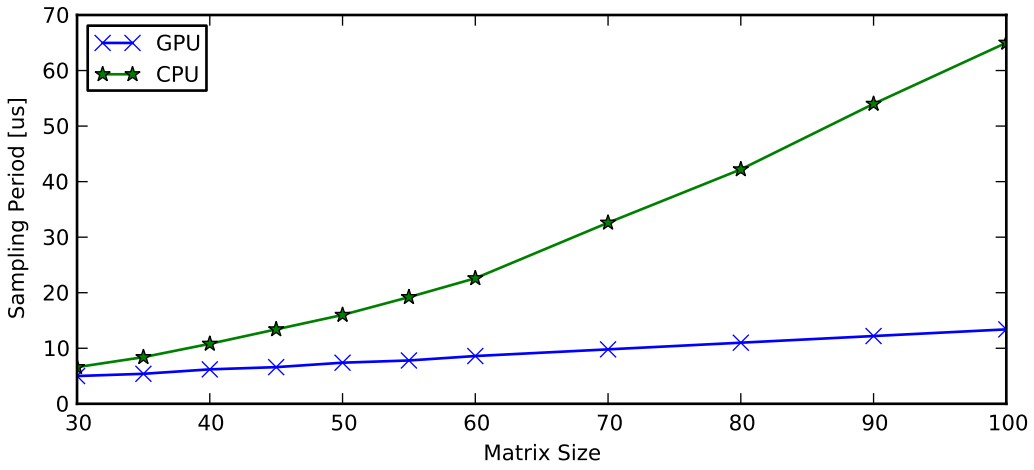

Figure 2.9: Lowest achievable sampling period for increasingly complex control computations (parametrized by matrix size). When running on the GPU, the achievable sampling periods are consistently lower in absolute numbers, and the scaling with increasing complexity is superior.

algorithm from the resulting gap in sample numbers.

Figure 2.9 compares the lowest achievable sampling periods for control algorithms of different complexity, parametrized by matrix size. This plot shows that, independent of the matrix size, the control computation was faster when implemented on the GPU. In addition to that, the scaling with increasing matrix size is more favorable for a GPU implementation as well, i.e. if the algorithm becomes more complex, the increase in sampling period is smaller for a GPU implementation than for the CPU implementation. For matrix sizes of less than 30 x 30, the sampling period reaches the lower bounds imposed by data transfer and AD/DA conversion, so that the time required for computation can no longer be measured.

It should be noted that Figure 2.9 is specific to the algorithm that is being tested, and a different control algorithm would give different results. However, with matrix multiplication being a very common operation in most control algorithms that typically dominates over all other required computations, Figure 2.9 nevertheless provides a representative benchmark.

Another point worth explaining is the different scaling of the CPU and GPU implementations: while the GPU implementation is linear, the CPU implementation is approximately quadratic. The reason for that is that the application of a n by n matrix to a vector requires roughly $2(n^2+n)$ operations which parallelize very well. While the CPU has to execute all these operations in sequence, the GPU implementation is able to distribute them to n different threads.

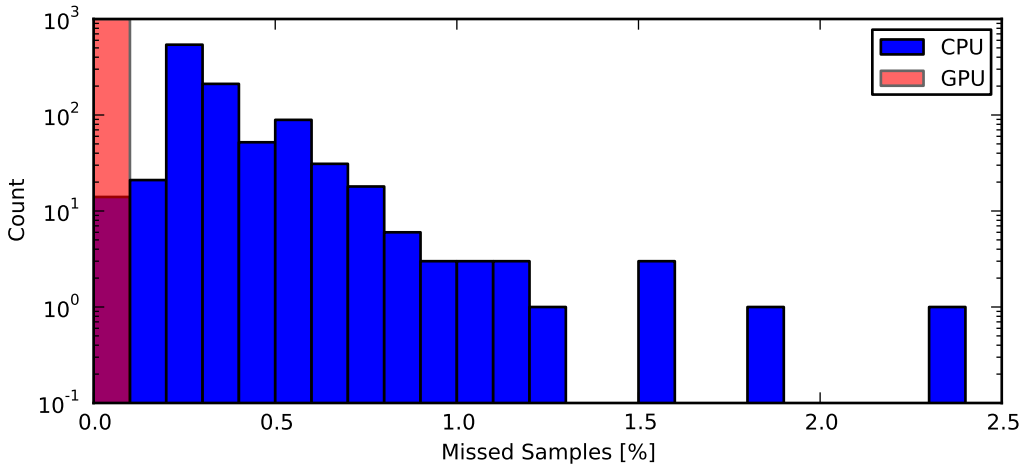

Figure 2.10: Lost samples in 1000 runs of the benchmark algorithm with a 35 by 35 matrix on GPU and CPU at minimum sampling interval. While the GPU is able to process all samples in every run, the CPU frequently has to skip over samples when execution of the algorithm is interrupted by the operating system.

Comparison with FPGA

When attempting to program the benchmark algorithm into HBT-EP's existing FPGA boards, it turned out that the boards had insufficient capacity to hold even the 30 by 30 matrices, making a direct comparison between GPU, CPU and FPGA impossible. However, in prior experiments [31], this system was used to implement a Kalman-filter-based algorithm with 2 internal states, 5 outputs and 5 inputs with a sampling period of 4 µs and latency of 12 µs. This algorithm pushed the FPGA boards to the limits of their capacity and required to work with integer rather than floating point numbers to conserve space. While FPGA capacity has improved since acquisition of these boards, it is safe to assume that a GPU-based system will at the very least be able to match the performance of current FPGA hardware, while being both cheaper to purchase and easier to program.

2.6.4 Real-Time Performance

A second important difference between the behavior of the GPU and CPU implementations of the benchmark algorithm presented in the last section is illustrated in Figure 2.10. To generate this figure, both the CPU and GPU implementations of the benchmark algorithm were run 1000 times, each time processing 10000 samples. The matrix size in all the runs was 35 by 35, and both CPU and GPU were run at the shortest possible sampling period (as

shown in Figure 2.9). While the GPU is always able to process all samples, the CPU frequently misses samples (which the algorithm detects by means of the sample number). This is because the run time of the CPU algorithm is not fully deterministic even when running with real-time priority and CPU cycles still have to be shared with the operating system. The GPU implementation, on the other hand, has exclusive access to the GPU cores and is thus fully deterministic.

2.7 Open-Loop Tests

HBT-EP's new GPU-based control system has been fully operational since June 2012. In addition to that, operation at reduced frequencies up to 100 kHz has been possible since September 2011. In this time, the control system has already been used extensively in open-loop, or "pre-programmed" mode. A detailed discussion of these experiments is beyond the scope of this thesis (which focuses on feedback control). However, I will provide a brief summary of the main use cases in order to illustrate the flexibility and versatility of the system.

Phase Flips of Resonant Magnetic Perturbations The control system was programmed to generate a helical field with a pitch matching the edge safety factor of the plasma, and the phase of the field was flipped. By correlating the amplitude of the generated currents with the amplitude of the measured field, the plasma response to the resonant magnetic perturbation was calculated. (Experiments led by Daisuke Shiraki).

Rotating Applied Perturbations In the absence of external control fields, HBT-EP plasmas are typically subject to rotating helical perturbations. To study these, the control system was programmed to generate rotating external fields with the same helicity as the naturally occurring one. The rotation frequency of the applied field was varied within a shot to examine the response of the plasma when the frequency of the applied field matched the frequency of the naturally rotating mode. (Experiments led by Qian Peng and Nikolaus Rath).

Axisymmetric Shaping The control system was used to generate axisymmetric, static perturbations to get an idea of the effects of a planned new shaping coil. (Experiments led by Patrick Byrne).

Mode Locking Threshold A naturally rotating perturbation can be stopped and "locked" in a specific orientation using external fields. The external field amplitude at which this happens gives information about the torque applied to the plasma. To determine this locking threshold amplitude, the control system was used to generate resonant fields with a linear amplitude ramp over the course of a shot. (Experiments led by Qian Peng).

Breakdown Radius Adjustment HBT-EP plasmas break down at a time when the vertical field coils are not yet active. Therefore, the major radius at which the breakdown begins is determined by the geometry of the ohmic heating coil and, in the past, could not easily be adjusted. The control system was used to generate a transient vertical field at the beginning of the shot to adjust the break-down radius for the time until the activation of the vertical field coils. (Experiments led by Nikolaus Rath)

Sensor Testing HBT-EP has a total of 216 magnetic sensors whose signals pass through several layers of wiring. Being able to use the control system to create a known, localized magnetic field in the vicinity of a specific sensor turned out to be an invaluable help in debugging sensor problems.

Chapter 3

Signal Separation

The control system and algorithm described in this thesis is designed to control magnetic perturbations from an axisymmetric equilibrium state. When determining the shape and amplitude of any such perturbations, it is thus necessary to separate sensor signals into contributions from the equilibrium fields and contributions from perturbations. This chapter explains how this separation is performed. Unless explicitly stated, all later chapters assume that equilibrium contributions have been subtracted from sensor signals with the techniques described here.

The proposed method is based on an application of singular value decomposition called biorthogonal decomposition (BD) that allows combining spatial and temporal information and does not require explicit knowledge of the equilibrium. In contrast to other techniques like signal smoothing, BD-based signal separation is able to correctly identify non-rotating perturbations and requires less caution in the choice of separation parameters.

3.1 The Identification Problem

Formally, the perturbation of the plasma from an axisymmetric equilibrium state can be determined by reconstructing the axisymmetric equilibrium and then subtracting the sensor signals that would be generated by this equilibrium. If the axisymmetric equilibrium is not known but a sufficient number of sensors are available, a theoretically straightforward alternative is to average all sensor measurements over the toroidal angle and treat this average as the equilibrium signal.

However, in practice both methods are often unfeasible because of sensor misalignments. Because of the order-of-magnitude differences between the equilibrium and perturbed signals, even a slight misalignment of a sensor can result in some components of the equilibrium field

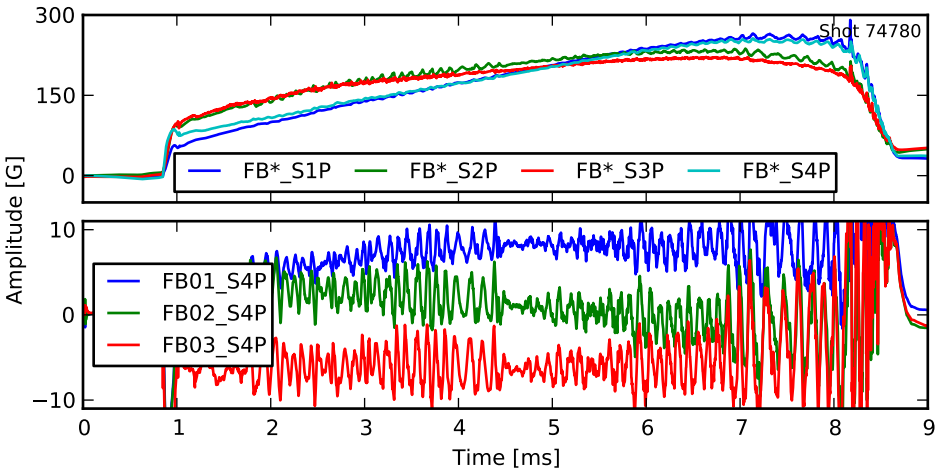

Figure 3.1: Top: toroidally averaged poloidal field measurements at 4 different poloidal locations. Bottom: difference between signal and toroidal average at 3 successive toroidal locations.

being treated as perturbations of a magnitude that completely shadows the true perturbed signal.

This problem is illustrated in Figure 3.1. In the upper plot, this figure shows the toroidally averaged signal at four poloidal locations for a typical HBT-EP shot. The signals have been obtained from the 4x10 grid of poloidal feedback sensors (plotted green in Figure 1.3). While the equilibrium signals look reasonable, the results for the perturbed signals obtained by subtracting the equilibrium signal from the individual signals are not plausible. The problem is illustrated in the lower plot of Figure 3.1, which shows the difference between full signal and toroidal average for three sensors spaced 36° apart toroidally. If this result is to be trusted, the plasma is subject to a constant perturbation with toroidal mode number 5, and a superimposed high frequency oscillation. This is physically unlikely, and it will be shown that other separation methods produce more plausible results.

3.2 Biorthogonal Decomposition

Biorthogonal decomposition (BD) [13, 1] is an application of singular value decomposition (SVD) to a matrix of spatially related rows and temporally related columns. The BD expands a set of time dependent measurements at different spatial locations as a sum of coherent modes without any a priori assumptions.

Consider a set of sensors x_i that take measurements at discrete time points t_j. Let the measurements performed by a single sensor at different times be written as a row vector, and the matrix formed by stacking the row vectors of the different sensors be $\mathbf{X}$, so that $X_{ij} = x_i(t_j)$. While not necessary for the BD in the general case, for the purpose of this thesis any measurements used for BD will always be zero centered first, so that $\sum_j x_i(t_j) = 0$.

In this case the SVD $\mathbf{U}.\mathbf{S}.\mathbf{V}^\dagger$ of $\mathbf{X}$ is called a biorthogonal decomposition, and the columns of $\mathbf{U}$ and $\mathbf{V}$ (or rows of $\mathbf{V}^\dagger$) are called spatial and temporal modes respectively.

The significance of these modes can be understood as follows. By the definition of the SVD, the columns of $\mathbf{U}$ are orthonormal and eigenvectors of $\mathbf{X}.\mathbf{X}^\dagger$ with eigenvalues $\sqrt{S_{ii}}$. Now consider the elements of the $\mathbf{X}.\mathbf{X}^\dagger$ matrix:

$$\left(\mathbf{X}.\mathbf{X}^\dagger\right)_{ij} = \sum_k x_i(t_k) x_j(t_k) \tag{3.1}$$

Since $\sum_j x_i(t_j) = 0$, the elements of $\mathbf{X}.\mathbf{X}^\dagger$ are thus the correlation coefficients between the different sensors. Consider now a hypothetical second set of sensors $y_i(t_j)$ where each sensors measures a linear combination of the real sensors:

$$y_i(t_j) = \sum_k U_{ki} x_k(t_j) \tag{3.2}$$

Since the columns of $\mathbf{U}$ are eigenvectors of $\mathbf{X}.\mathbf{X}^\dagger$, the fictitious sensors y_i will be uncorrelated in time. Also, by virtue of being linear combination of the real sensors, every sensor y_i measures a spatially distributed structure that has amplitude U_{ji} at the position of the real sensor j. The BD thus allows understanding the measurements as being generated by the independent time evolution of different static spatial structures – the spatial modes. The singular value of a mode is proportional to the overall amplitude of the mode and thus gives a measure of how much a given mode affects the measurements over time.

In order to understand the temporal modes, a similar argument can be made using the matrix $\mathbf{X}^\dagger.\mathbf{X}$. An element at row i and column j of this matrix is the correlation coefficient between the spatial structure (determined by the values of all sensors at an instant in time) at time t_i and t_j. The eigenvectors of $\mathbf{X}^\dagger.\mathbf{X}$ correspondingly describe a superposition of measurements in time that are uncorrelated in space. However, this point of view does not give a similarly meaningful interpretation for the way that measurements are generated, so that a temporal mode i is typically interpreted as the evolution of the amplitude of the spatial

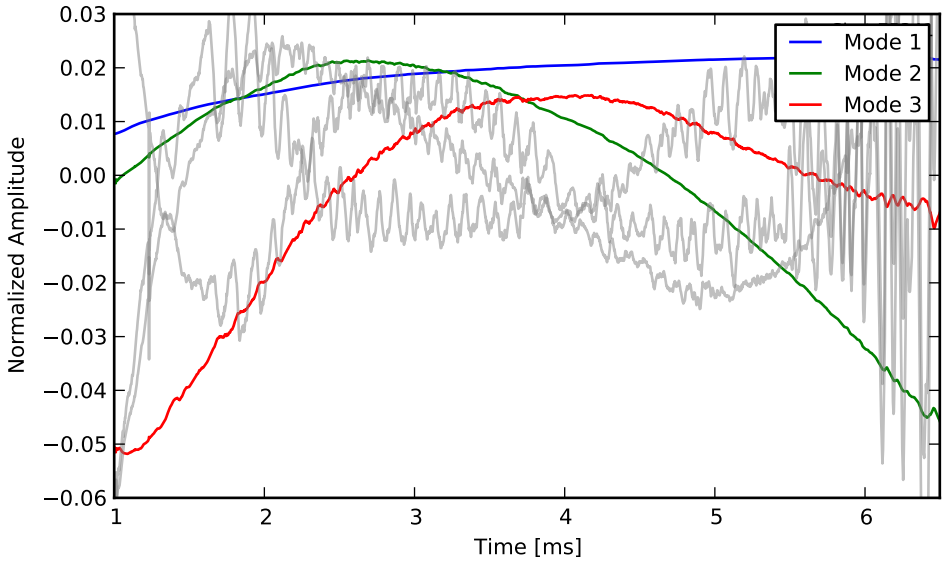

Figure 3.2: Temporal evolution of the first 6 most significant modes after biorthogonal decomposition. The three most significant modes are expected to contain the equilibrium components of the signal and are plotted in color.

mode i.

3.3 Equilibrium Modes

As explained above, the biorthogonal decomposition decomposes a set of measurements from different points in space and time into a superposition of spatially invariant and temporally uncorrelated modes. This makes it very well suited to compute the equilibrium components in a set of sensor signals: even if the individual sensors are not perfectly aligned, signals from the equilibrium field must result in signals that are spatially almost rigid and change comparatively slowly in time. This means that those structures have a very high chance of being isolated as individual modes when performing a biorthogonal decomposition *over the entire plasma lifetime*. A method to determine the equilibrium components in a set of sensor measurements is therefore to decompose the measurements into spatial and temporal modes using biorthogonal decomposition, identify the modes corresponding to equilibrium signals, and then reconstruct the signals from just these modes.

There are two criteria that allow classification of BD modes as equilibrium or perturbed modes. Generally, equilibrium signals have higher amplitudes, so equilibrium modes are

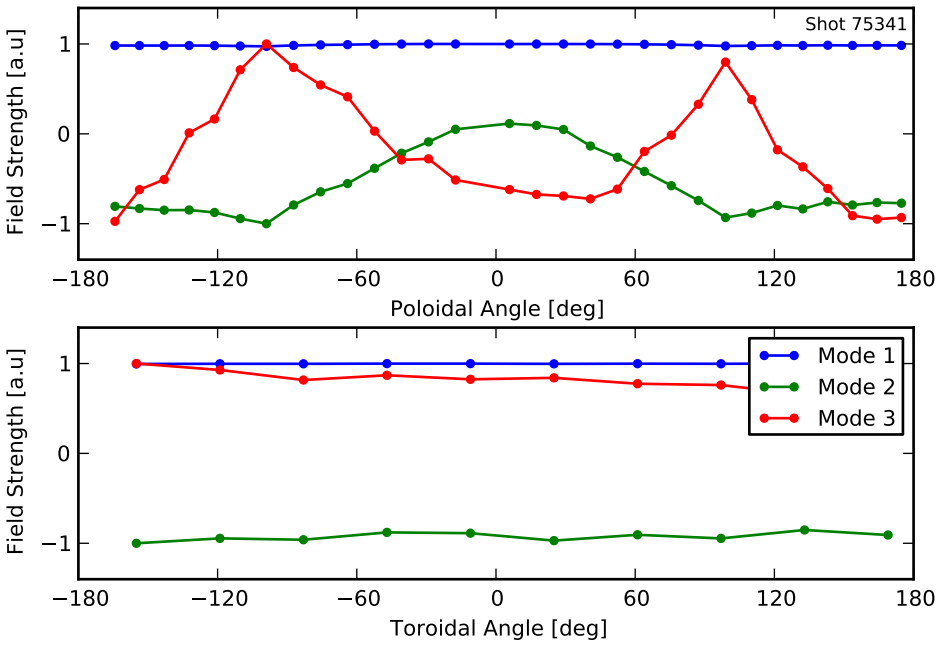

Figure 3.3: Spatial profiles of the equilibrium modes as determined by biorthogonal decomposition. Plotted are the amplitudes of the poloidal sensors in a poloidal array (PA1_SxxP, upper plot) and a toroidal array (FBxx_S1P, lower plot).

expected to have larger singular values than perturbed modes. If BD modes are analyzed in order of decreasing singular values, the search for equilibrium modes can therefore be stopped as soon as the first perturbed mode is found. The first identification criterion is the frequency spectrum of the temporal mode. Equilibrium modes will have most of their amplitude concentrated in the frequencies over which the equilibrium currents change, i.e. in on a comparatively slow timescale. The second criterion is the toroidal variation of the mode. The toroidal variation of an equilibrium mode is expected to be due to sensor misalignments only, and thus restricted to a few percent. Perturbed modes, on the other hand, may not be axisymmetric and thus have toroidal variations as large as the individual sensor signals.

Once the equilibrium modes have been identified, reconstruction of the equilibrium signal components is trivial by setting the singular values of perturbed modes to zero in $\mathbf{S}$ and then carrying out the matrix multiplication $\mathbf{U}.\mathbf{S}.\mathbf{V}^{\dagger}$ to obtain the measurement matrix $\mathbf{X}$ with all perturbed components eliminated.

When applied to HBT-EP plasmas (using a 20% threshold for toroidal variation, or requiring 87% of the amplitude in the temporal mode to be in frequencies below 1 kHz), both classification methods give very similar results and identify the first three or four BD modes as equilibrium modes. In the case where the methods differ, the 4rd mode generally has a much smaller singular value than the 3rd mode and thus does not significantly affect the reconstructed equilibrium modes. The number of equilibrium modes is not surprising: in HBT-EP, the equilibrium fields are governed by three independent equilibrium coils: the vertical field coil, the ohmic heating coil, and the toroidal field coil. These three independent parameters are therefore likely to result in three spatial equilibrium modes.

Figure 3.2 shows the temporal evolution of the 6 most significant modes as determined by biorthogonal decomposition of all available magnetic sensor signals in a typical HBT-EP shot. The first three modes have been identified as equilibrium modes, and their spatial profiles are shown in Figure 3.3. As expected, the temporal evolution of the equilibrium modes differs significantly from the remaining modes, and their spatial structure shows almost no toroidal variation. The poloidal variations correspond to uniform horizontal displacement and vertical stretching and compression.

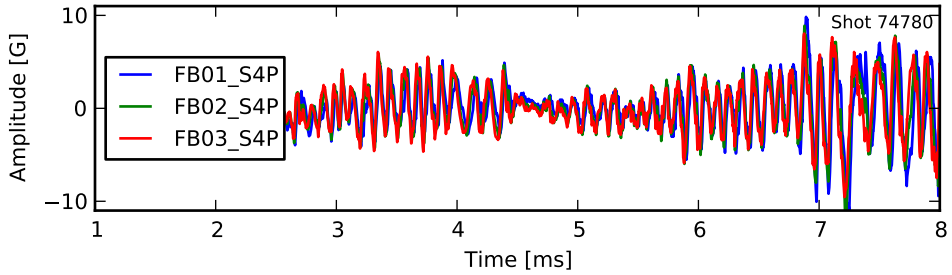

Figure 3.4: Difference between full signal and BD-determined equilibrium signal at 3 successive toroidal locations. In contrast to Figure 3.1, this difference is much more likely to be the perturbed signal as it represents a helical, toroidally rotating mode.

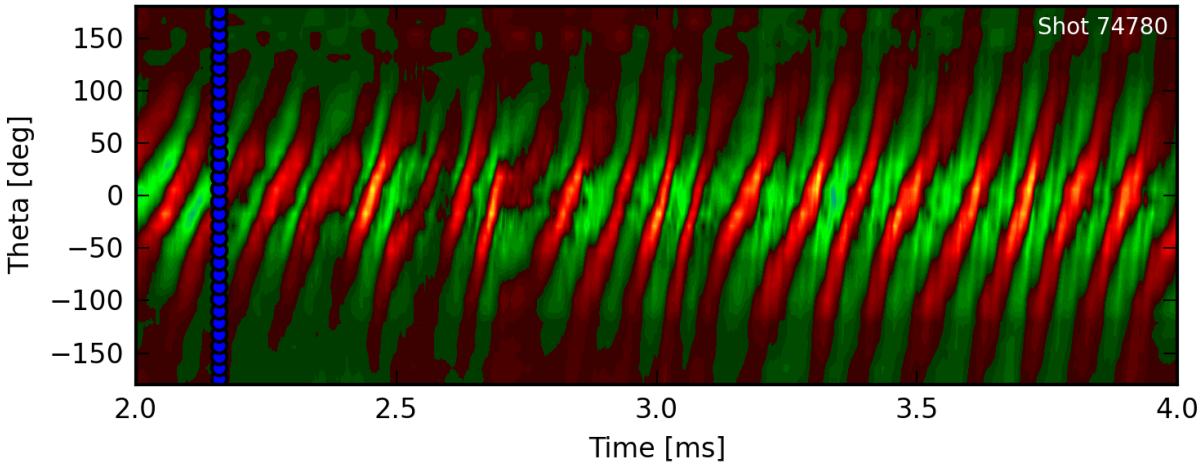

Figure 3.5: Perturbation amplitudes in a poloidal sensor array. The fluctuations are in the order of 12 Gauss. The equilibrium components of the signals have been subtracted using biorthogonal decomposition as described in Section 3.2. Blue dots represent sensor locations.

3.4 Perturbation Structure

Figure 3.4 shows the perturbed signals when they are calculated as the difference between the BD-determined equilibrium modes and the full signals, and should be contrasted with the lower plot in Figure 3.1. With BD-based separation, the amplitude at the three toroidal locations is very similar, and each signal oscillates around zero with a slightly different phase, indicating a toroidally rotating structure. An even better picture can be gained by plotting the amplitudes for all sensors at a given toroidal or poloidal location over time as done in Figure 3.5. Here one can clearly see that the perturbations take the form of a coherent, rotating modes.

By performing a second biorthogonal decomposition on the perturbed signals and over smaller time windows, the perturbed signals can be further separated into multiple independent modes with a fixed toroidal mode number that rotate toroidally with frequencies from 3 to 14 kHz. The observed poloidal spectrum of these modes was found to depend on the plasma major radius, and the amplitude of a given mode typically increases dramatically when the plasma edge safety factor coincides with the ratio of the dominant poloidal to the toroidal mode number. It is therefore hypothesized that these perturbations are saturated, current-driven external kink modes interacting with HBT-EP's resistive shell segments, i.e. resistive wall modes.

Figure 3.6 shows a close up of an example signal from a single poloidal field sensor together with the equilibrium signal reconstructed from the BD modes. As expected, the equilibrium signal resembles the overall evolution of the total signal but short term fluctuations have been filtered out.

3.5 Temporal Smoothing

A second way to identify equilibrium signals is to perform temporal smoothing on individual sensor traces. The idea behind this can be seen in Figure 3.6: as long as perturbations rotate, the resulting high frequency oscillations in individual sensor signals will be smoothed out, leaving the equilibrium signal. This approach has been used successfully in past work [56, 40, 44]. Here, biorthogonal decomposition was used only as a second step after temporal smoothing to determine the shape and time evolution of the perturbations. A temporal smoothing separation algorithm will also be used by the control algorithm described in Chapter 4, because the BD-based separation requires knowledge of the past and future of

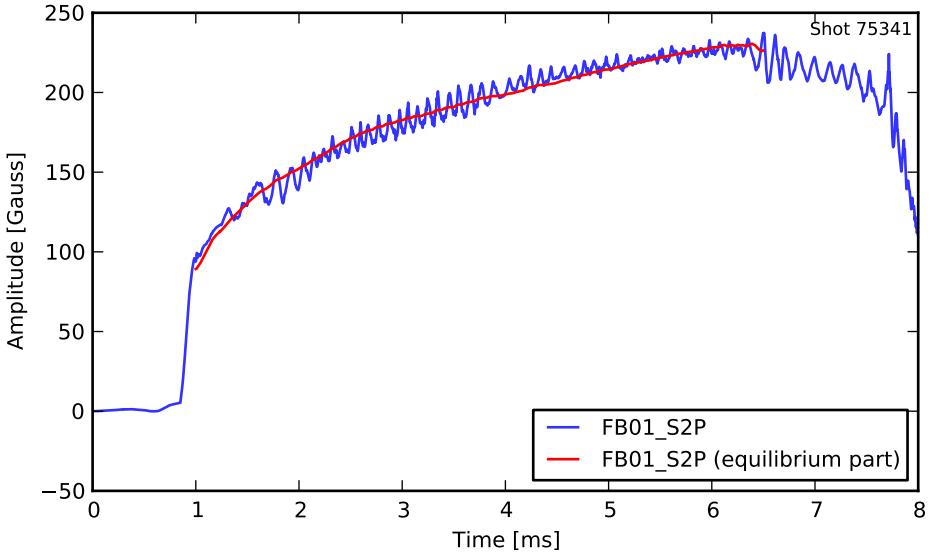

Figure 3.6: A typical poloidal sensor signal in an HBT-EP discharge and its equilibrium component as determined by biorthogonal decomposition.

a signal and is thus not suitable for real-time control. However, the BD-based separation method has several advantages over a pure temporal smoothing:

- BD-based separation combines spatial with temporal information. While temporal smoothing works separately on every sensor signal and ignores any spatial information, BD-based separation integrates measurements from all sensors at all times.

- BD-based separation has fewer adjustable parameters, and is less sensitive to their values. Nominally, the classification of equilibrium modes has three parameters (permissible toroidal variation, frequency threshold and permissible amplitude in frequencies above the threshold), but in practice a change of these parameters within reasonable boundaries does not result in changes of the computed equilibrium signals. Temporal smoothing, on the other hand, allows for a large range of smoothing techniques and parameters, which have been shown to result in different equilibrium signals.

- BD-based separation is able to correctly identify *locked modes*, i.e. perturbations with zero frequency. Every temporal smoothing method will necessarily treat locked modes as part of the equilibrium signals, because their toroidal variation becomes invisible when sensors are only considered individually.

BD-based signal separation has therefore been used for all post-shot analysis presented in this thesis.

Chapter 4

Control Algorithm

This chapter presents an algorithm for adaptive control of rotating magnetic perturbations that is designed to take full advantage of the computational resources provided by a GPU-based control processor. The distinctive feature of this algorithm is that it is non-linear and *adaptive*, i.e. the system model used by the control algorithm is time dependent. For each rotating perturbation, both frequency and growth rate are continuously derived from the time evolution of the perturbation itself and then used to predict its future evolution, filter out noise, and compensate for frequency dependent transfers functions of the control coils and output amplifiers. Non-linear computations are required for the translation between cosine and sine amplitudes and the corresponding toroidal phase and quadrature-pair amplitude in both the interpretation of measurements and the generation of control signals.

4.1 System Model

Consider an arbitrary rotating magnetic perturbation that produces fields $\vec{B}(\phi,\theta,r,t)$ at the location with toroidal angle ϕ, poloidal angle θ and minor radius r. Since the perturbation is rotating with rigid structure, it must satisfy

$$\vec{B}(\phi,\theta,r,t) = e^{\gamma(t)t}\,\vec{B}(\phi+\delta(t),\theta,r,0) \tag{4.1}$$

where $d\delta/dt$ is the (possibly time varying) rotation frequency and γ the (possible time varying) growth rate. Note that even though $e^{\gamma(t)t}$ suggests an exponential growth or decay, the above equation allows any time evolution $f(t)$, since γ may be chosen as $\gamma(t) = \log[f(t)]/t$. The

spatial structure of the perturbation may be expanded in a Fourier series,

$$\vec{B}(\phi,\theta,r,0) = \sum_n \vec{a}_n(\theta,r)\, e^{in\phi} \tag{4.2}$$

so that we may define the real coefficients $A_\mathrm{c}(t)$ and $A_\mathrm{s}(t)$ as

$$\begin{aligned}\vec{B}(\phi,\theta,r,t) &= e^{\gamma(t)t} \sum_n \vec{a}_n(\theta,r)\, e^{in\phi} e^{in\delta(t)} \\ &=: \sum_n \vec{a}_n(\theta,r) e^{in\phi} \left(A_n^{(\mathrm{c})}(t) + i A_n^{(\mathrm{s})}(t)\right)\end{aligned} \tag{4.3}$$

The time evolution of any rotating rigid mode is thus determined by the functions $\gamma(t)$ and $\delta(t)$, and its spatial structure defined by the functions $\vec{a}_n(\theta,r)$.

The time evolution of a perturbation is thus assumed to be described by the equations

$$A_n^{(\mathrm{c})}(t) = A_n^{(\mathrm{c})}(0)\, e^{\gamma(t)t} \cos\big(n\delta(t)\big) \tag{4.4}$$

$$A_n^{(\mathrm{s})}(t) = A_n^{(\mathrm{s})}(0)\, e^{\gamma(t)t} \sin\big(n\delta(t)\big) \tag{4.5}$$

Note that no explicit assumptions about the effects of control signals are made, since they are already contained in the free functions $\gamma(t)$ and $\delta(t)$.

Both the Boozer and Fitzpatrick-Aydemir plasma models that will be used for simulations in Chapter 8 describe evolutions that fit this model. For these models, the free functions take the form

$$\gamma(t) = \gamma_0 \tag{4.6}$$

$$\delta(t) = \delta_0 + \omega_0 t \tag{4.7}$$

for fixed values of γ_0, ω_0 and δ_0.

For simplicity, the following explanations assume that only one rigid perturbation is tracked. However, in practice $A_n^{(\mathrm{c})}$, $A_n^{(\mathrm{s})}$, $\gamma(t)$ and $\delta(t)$ each carry an additional index that enumerates the different perturbations that the control system tracks.

The main assumption that goes into the design of the system model and control algorithm is that that there exists an intermediate time scale T such that for any t_0,

$$A_n^{(\mathrm{c})}(t : |t-t_0| < T) \approx e^{\gamma(t_0)t} \cos\big(n\omega(t_0)t + n\delta(t_0)\big) \tag{4.8}$$

$$A_n^{(\mathrm{s})}(t : |t - t_0| < T) \approx e^{\gamma(t_0)\,t} \sin\!\big(n\omega(t_0)\,t + n\delta(t_0)\big) \tag{4.9}$$

where

$$\omega(t) \coloneqq \frac{d\delta}{dt} \tag{4.10}$$

For every rigid perturbation, the control algorithm keeps track of the coefficients $A_n^{(\mathrm{c})}(t)$ and $A_n^{(\mathrm{s})}(t)$, the instantaneous rotation frequency ω, and the instantaneous growth rate γ.

4.2 Measurement

Sensors are assumed to sample the perturbed field $\vec{\Phi}$ at discrete positions. Let the sensor positions be ϕ_i, θ_i, r_i with normal vectors $\vec{n}_i$. The field $\Phi_i^{(\mathrm{s})}$ measured by the i-th sensor is then given by

$$\Phi_i^{(\mathrm{s})} = \sum_n \big(\vec{a}(\theta_i, r_i) \cdot \vec{n}_i\big)\, e^{in\phi_i} \left(A_n^{(\mathrm{c})}(t) + i\, A_n^{(\mathrm{s})}(t)\right) \tag{4.11}$$

By collecting all measurements into a vector $\vec{\Phi}_{\mathrm{s}}(t)$, and the coefficients $A_n^{(\mathrm{c})}$ and $A_n^{(\mathrm{s})}$ into vectors $\vec{A}_{\mathrm{c}}$ and $\vec{A}_{\mathrm{s}}$, Equation 4.11 becomes a matrix equation

$$\vec{\Phi}_{\mathrm{s}} =: \mathbf{C}_{\mathrm{c}}.\vec{A}_{\mathrm{c}} + \mathbf{C}_{\mathrm{s}}.\vec{A}_{\mathrm{s}} \tag{4.12}$$

In order to obtain the coefficients $\vec{A}_{\mathrm{c}}$ and A_{s} required by the system model, the control algorithm computes the pseudoinverses $\mathbf{C}_{\mathrm{c}}^{-1}$ and $\mathbf{C}_{\mathrm{s}}^{-1}$ and applies them to the measurement vector $\vec{\Phi}_{\mathrm{s}}$. (When tracking multiple perturbations, they will increase the dimension of the $\mathbf{C}_{\mathrm{c}}$ and $\mathbf{C}_{\mathrm{s}}$ matrices but do not additional complexity.)

4.3 State Observation

The defining feature of the adaptive system model is that $\gamma(t)$ and $\omega(t)$ are not set a priori, but derived from the time evolution of the measured amplitudes $A_n^{(\mathrm{c})}(t)$ and $A_n^{(\mathrm{s})}(t)$. For every such measurement, the system computes the instantaneous toroidal phase $\delta(t)$ and quadrature amplitude $A(t)$ as

$$A(t) = \sqrt{\sum_n \big(A_n^{(\mathrm{c})}\big)^2 + \sum_n \big(A_n^{(\mathrm{s})}\big)^2} \tag{4.13a}$$

$$\delta(t) = \frac{1}{N}\sum_n \arctan\left(A_n^{(\mathrm{s})}, A_n^{(\mathrm{c})}\right) \tag{4.13b}$$

where N is the number of expansion terms in Equation 4.2. Based on this, the system model parameters are continuously calculated as the least squares solutions to the phase rotating with constant frequency $\omega(t_0)$, and the quadrature amplitude growing (or decaying) exponentially with fixed time constant $\gamma(t_0)$:

$$e^{\gamma(t_0)\,t} \approx A(t)/A(0) \tag{4.14a}$$

$$\delta_0 + \omega(t_0)\,t \approx \delta(t) \tag{4.14b}$$

This fitting is to be performed over the moving time window $t_0 - T < t < t_0$, resulting in a slow evolution of $\gamma(t)$ and $\omega(t)$.

Expressed in closed form, the system state at time t thus always satisfies the following equation

$$A_n^{(\mathrm{c})}(t) = \mathbf{C}_\mathrm{c}^{-1}.\vec{\Phi}_\mathrm{s}(t) \tag{4.15a}$$

$$A_n^{(\mathrm{s})}(t) = \mathbf{C}_\mathrm{s}^{-1}.\vec{\Phi}_\mathrm{s}(t) \tag{4.15b}$$

$$\gamma(t) = \int_{t-T}^{t} c(\tau - t)\,\log\sqrt{\sum_n \left(A_n^{(\mathrm{c})}(\tau)\right)^2 + \sum_n \left(A_n^{(\mathrm{s})}(\tau)\right)^2}\,d\tau \tag{4.15c}$$

$$\omega(t) = \int_{t-T}^{t} \sum_n \frac{c(\tau - t)}{N}\,\arctan\left(A_n^{(\mathrm{s})}(\tau), A_n^{(\mathrm{c})}(\tau)\right)\,d\tau \tag{4.15d}$$

were $c(t)$ are the least squares fitting coefficients. Since γ and ω are part of both the system model and the system state, this set of equation describes a system with a time-varying model that *adapts* to the measured dynamics.

It should also be noted that when using these equations directly, the control system would have to iterate over the last N samples for every processed sample. However, as will be explained in Section 5.3, the equations can be rewritten in such a way that the control system can update γ and ω continuously, using just the information about samples at time t and $t - T$ at every step.

4.4 Control Signal Generation

The basic assumption for the generation of control output is that a magnetic perturbation can be controlled if the control coils can generate a similarly shaped field with any desired amplitude and phase difference.

For a set of control coils located at ϕ_i, θ_i, r_i with normal vectors $\vec{n}_i$, the perturbed field at these points is given by Equation 4.3. As for the measurements, the field can be expressed in terms of matrices acting on coefficient vectors $\vec{A}_\mathrm{c}$ and $\vec{A}_\mathrm{s}$. Denoting these matrices with $\mathbf{F}_\mathrm{c}$ and $\mathbf{F}_\mathrm{s}$, the control coil current configuration $\vec{I}$ most suitable to *amplify* a perturbation with coefficients $\vec{A}_\mathrm{c}$ and $\vec{A}_\mathrm{s}$ is thus

$$\vec{I}(t) = \mathbf{F}_\mathrm{c}.\vec{A}_\mathrm{c} + \mathbf{F}_\mathrm{s}.\vec{A}_\mathrm{s} \tag{4.16}$$

For the generation of control output, it is more useful to express the required currents in terms of the toroidal phase $\delta(t)$ and quadrature amplitude $A(t)$ of the perturbation:

$$I_m(t) = A(t) \sum_n \left[F^{(\mathrm{c})}_{mn} \cos\big(n\delta(t)\big) + F^{(\mathrm{s})}_{mn} \sin\big(n\delta(t)\big) \right] \tag{4.17}$$

In this form, it is possible to manipulate the $\delta(t)$ and $A(t)$ for the desired feedback gain and phase, and to compensate for noise, digital latency and analog response.

Let the control system have a latency of τ_lat, and a sampling period of τ_s. The processing of a sample k digitized at time $t = k\tau_\mathrm{s}$ begins with initialization of $A[k]$ and $\delta[k]$ from the measurements $\vec{\Phi}_\mathrm{s}(t)$ using Equations 4.12 and 4.13. Both $A[k]$ and $\delta[k]$ are then transformed in multiple steps. A closed expression that includes all the transformations is very convoluted, so the rest of this section will use an imperative, sequential notation of the form $a \leftarrow f(a)$ to define the different processing steps. This should be read like an expression in an imperative programming language: the value of a is changed from a to $f(a)$.

4.4.1 Filtering

The first step in the generation of the control output is designed to reduce the effects of stochastic measurement noise. Similar to the techniques employed by a Kalman filter, the control output is not based on the instantaneous values of $A[k]$ and $\delta[k]$ but on a weighted average of a prediction based on the system model and the most recently measured value. The prediction is made by advancing the previous measurement by the sampling interval τ_s

using the fitted growth rate γ and rotation frequency ω:

$$A[k] \leftarrow \alpha A[k] + (1-\alpha)\, A[k-1]\, e^{\gamma[k]\tau_s} \tag{4.18}$$

$$\delta[k] \leftarrow \beta\delta[k] + (1-\beta)\,(\delta[k-1] + \omega[k]\tau_s) \tag{4.19}$$

Here α and β are parameters between zero and one that characterize the degree of filtering. With $\alpha = \beta = 1$, no filtering takes place and the output is based only on the most recent measurement. However, both rotation frequency and growth rate are still used to compensate for latency and coil response in later steps.

4.4.2 Gain and Phase

The next step is to apply a feedback gain g and base feedback phase $\Delta\phi$. Ideally, a control phase of 0° will result in positive, and a control phase of 180° in negative feedback. The feedback gain is essentially a conversion from the units of the measured perturbation to units suitable for the control output. This transformation takes the simple form

$$A[k] \leftarrow g \cdot A[k] \tag{4.20}$$

$$\delta[k] \leftarrow \delta[k] + \Delta\phi \tag{4.21}$$

In practice, there is often a phase shift between sensors and control coils because one only defines the normal component of the perturbation $\vec{B}$. In that case, there is an unknown phase difference between the mode measured by the input matrices $\mathbf{C}_{c,s}$ and generated by the output matrices $\mathbf{F}_{c,s}$ (cf. Equation 4.16), which needs to be incorporated into $\Delta\phi$. This offset is typically determined by a scan over $\Delta\phi$.

4.4.3 Latency Compensation

In the third step, the control algorithm compensates for the system latency τ_{lat}. The control output for a perturbation measured at time t will only be produced at time $t+\tau_{lat}$. In order to compensate for this, the phase and gain of the control output needs to be adjusted, which is done based on the expected changes of $A[k]$ and $\delta[k]$. In a time τ_{lat}, the perturbation grows

by roughly $e^{\gamma[k]\tau_{\text{lat}}}$ and rotates by roughly $\omega[k]\tau_{\text{lat}}$, so the appropriate transformation is

$$A[k] \leftarrow A[k]e^{\tau_{\text{lat}}\gamma[k]} \tag{4.22}$$

$$\delta[k] \leftarrow \delta[k] + \tau_{\text{lat}}\omega[k] \tag{4.23}$$

4.4.4 Analog Response Compensation

In a typical setup, the control system generates a voltage signal that controls amplifiers which try to establish a corresponding current in a set of control coils. However, inductances and resistances in both control coils and amplifiers will cause additional frequency dependent phase shifts and gain changes in the produced field. For a sinusoidal voltage signal with frequency ω and amplitude A, the resulting magnetic field will thus have the same frequency, but a different phase $\Delta\phi(\omega)$ and amplitude $\eta(\omega)A$.

The last step before the final control output therefore compensates for such frequency dependent responses in the analog control system components. Both phase and gain response is modeled as a polynomial in the rotation frequency, so that

$$\Delta\phi(\omega) = \sum_i H_i\omega^i \tag{4.24}$$

$$\frac{1}{\eta(\omega)} = \sum_i G_i\omega^i \tag{4.25}$$

The coefficients H_i and G_i completely specify the analog response and must be measured experimentally or calculated from knowledge of the analog circuits.

With the coefficients determined, the analog response is compensated for by the transformation

$$A[k] \leftarrow A[k] \cdot \sum_i G_i\gamma[k]^i \tag{4.26}$$

$$\delta[k] \leftarrow \delta[k] - \sum_i H_i\omega[k]^i \tag{4.27}$$

4.5 Equilibrium Subtraction

The biorthogonal-decomposition-based signal separation method described in Chapter 3 can not be used by a real-time control system, because it requires knowledge of the entire

time-evolution of the signal. In order to allow real-time separation, a different "polynomial fitting" method was developed that depends only on past data.

The idea of the polynomial fitting separation method is the observation that the temporal evolution of the BD equilibrium modes can be well approximated by low order polynomials. Rotating perturbations to the equilibrium, on the other hand, will show up as quadrature mode pairs that each oscillate around zero with the rotation frequency of the perturbation. Therefore, one can expect that a least square fit of the signals to low-order polynomials over a moving window will remove the perturbed components if the fitted region includes at least one period of the slowest expected perturbation. Mathematically, the equilibrium component $f[i]$ of a signal $g[i]$ calculated over a time window with N samples is

$$f[i] = \sum_{n=0}^{N-1} a[n]g[i-n] \tag{4.28}$$

The coefficients $a[n]$ are calculated as follows. First consider the standard $N \times p$ least squares matrix $\mathbf{M}$ for fitting a set of data points to a polynomial of degree p:

$$M_{ij} = (i-1)^{j-1} \tag{4.29}$$

When applying the pseudoinverse $\mathbf{M}^{-1}$ to a set of data points, the resulting vector has the coefficients of the fitted polynomial as its elements. Therefore, the value of this polynomial at the current sample is given by coefficients $a[n]$ with

$$a[n] = \sum_{i=0}^{p-1} (\mathbf{M}^{-1})_{p-i,n} N^i \tag{4.30}$$

Figure 4.1 shows a comparison between a full signal, its equilibrium component as determined by BD, and its equilibrium component as determined by polynomial fitting with optimized degree and fitting window. Window and degree were chosen to minimize the root mean square of the difference to the results obtained using biorthogonal decomposition. In Figure 4.1, the window length was 540 µs and polynomial was linear. Comparison of the window length and polynomial order over multiple HBT-EP shots show that the optimal degree is almost always linear, and the optimal window length varies by only a few percent. As far as accuracy is concerned, the polynomial fitting method can therefore be considered a good replacement for BD in a real-time control system.

A straightforward implementation of the polynomial filter has a significant drawback. For

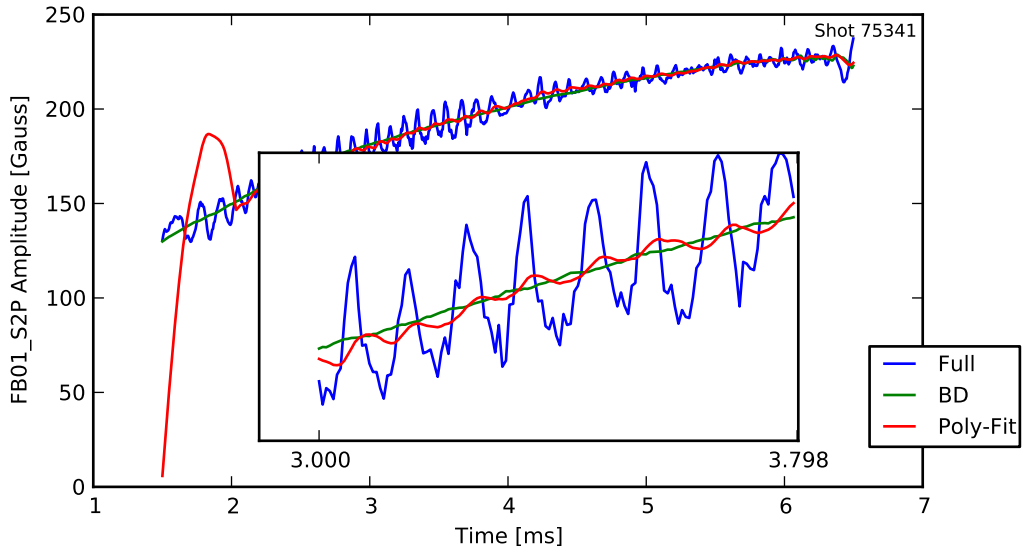

Figure 4.1: Comparison between a full sensor signal, its equilibrium components determined by BD and its equilibrium components determined by optimized polynomial fitting.

every sample that is to be processed, the last n samples have to be fitted to a new polynomial before the equilibrium amplitude at the current instant can be determined. This procedure thus results in a filter with as many coefficients as there are samples in the fitting window, which can generally not be applied quickly enough for real-time operation. However, this problem will be addressed in Section 5.3.

4.6 Summary of Parameters

In the description of the algorithm, there are several free parameters that are specific to the experimental situation. These are:

- The degree of the polynomial that is used to separate the full sensor signals into perturbed and equilibrium components, and the length of the time window over which this polynomial is fitted.
- The $\mathbf{C}_c$ and $\mathbf{C}_s$ matrices that define the shape of the perturbations that are to be controlled via the signals that they produce in the magnetic sensors.

- The lengths of the time windows over which the toroidal phase and quadrature amplitude of each perturbation is fit to a rotation frequency and growth rate.
- The coefficients α and β that define how much smoothing is applied to the quadrature amplitude and rotation frequency of each perturbation.
- The base feedback gain and feedback phase.
- The coefficients G_i and H_i that define the frequency dependent response of the analog actuators.
- The control system sampling interval and digital latency.
- The $\mathbf{F}_c$ and $\mathbf{F}_s$ matrices that define the shape of the perturbations that are to be controlled via the signals that would be produced if the control coils were used as sensors.

Chapter 6 will describe how these parameters have been determined for the HBT-EP tokamak.

Chapter 5

Implementation

This chapter describes the implementation of the adaptive control algorithm described in Chapter 4 for a GPU-based control system with the architecture presented in Chapter 2.

The implementation is contained in three separate modules. A preprocessor written in Python performs all calculations that do not depend on real-time control input. This is mostly the calculation of the various matrices and filter coefficients. The preprocessor writes a C header file that makes the results available to a loader and a kernel module. The loader is written in C++ and performs the control system initialization. It runs on the CPU and is responsible to set up memory buffers, load the kernel program code into the GPU, and start its execution. The kernel module implements the actual real-time control logic. It is written in C with the restrictions and extensions defined by the CUDA standard for GPU code.

The most time consuming calculations that need to be performed by the GPU kernel are matrix applications; either to perform least square fits over potentially long time windows, or to convert between input/output signals and perturbation amplitudes and phases. Both cases are therefore heavily optimized. Least squares problems are reformulated so that they can be solved continuously, and thread and memory layout are optimized for the structure of the measurement and control matrices.

5.1 Preprocessor and Loader

The first component of the control system software is a pre-processor written in Python. The pre-processor performs all computations that can be done ahead of time, and needs to be re-run only when some feedback parameters have been changed.

Since these computations are not time critical, the implementation is straightforward and utilizes standard numerical computing libraries to a large extent. The most important

quantities that are calculated by the preprocessor are the $\mathbf{C}_c$, $\mathbf{C}_s$, $\mathbf{F}_c$ and $\mathbf{F}_s$ matrices that define the shape of the modes that are to be controlled, and the coefficients to perform least square fittings over the different time windows. The preprocessor writes the results into a C header file that is included by the loader and GPU kernel.

The second component of the control system software is called the loader. The loader is written in C++ and responsible to initialize the control system and then pass control to the GPU kernel that runs the actual control algorithm. The standard interface for communicating with NVIDIA GPUs is the CUDA programming library provided by NVIDIA. However, until October 2012 the CUDA API did not offer a method to map device memory into the PCI-E address space as required to avoid redundant transfers of data through host memory. The NVIDIA GPU *drivers*, however, have provided the required functionality for several years. The control system was therefore developed using a reverse engineered library called *envyrt* instead of the CUDA library. Envyrt has been developed by PathScale for their ENZO GPU compiler suite, and offers only a limited set of features. However, it has all the functions that are required by the loader. It should be noted that envyrt is a replacement for the CUDA library, but still uses the NVIDIA GPU driver to communicate with the hardware.

The loader reads the compiled GPU kernel from a file and loads it into the GPU using envyrt. It then sets up the required input and output buffers, and communicates their addresses to the D-TACQ cards. Communication with these cards is done directly via *ioctl* calls on the device nodes. The cards are configured to wait for a trigger signal to arrive over an external input, and to then push and pull data from the GPU buffers at the configured sampling interval. After the cards have been initialized, the loader instructs the GPU to start running the loaded GPU kernel, and then waits for kernel execution to complete.

5.2 GPU Kernel

The GPU kernel is the most complex and interesting part of the control system software, and will be discussed in the rest of this chapter. The kernel is written in C with the extensions and restrictions defined by the CUDA standard. The full kernel source is included in Appendix C.

The GPU kernel has to be compiled into GPU code before it can be used by the loader. The choice of compiler is independent of which library is being used by the loader, and the system supports compilation with both NVIDIA's NVCC and PathScale's ENZO compiler.

The main challenge in the design of the GPU kernel is that all functionality needs to

be written from scratch. While there are extensive libraries available that provide GPU implementations for many numerical algorithms, all of these are designed as "one shot" functions. This means that they expect to be called by a CPU thread, and return control to the CPU when the solution has been computed. Since the control algorithm has to perform new least squares fittings and matrix applications every few microseconds for every new sample, passing control back and forth in this way is not feasible, and the necessary functions have to be written from scratch. The rest of this chapter illustrates some of the design choices that were made to optimize performance of the implementation.

Overall, the GPU kernel is implemented as a loop over the total number of samples that should be processed. At the beginning of this outer loop, an inner loop polls the input buffer until it detects that the A-D converter has pushed a new set of sensor samples into the input buffer. At the end of the loop, the computed control signals are written into the output buffer. The output buffer is periodically read by the D-A converter, but the GPU kernel has no notion of when this happens.

5.3 Continuous Least Squares Fitting

Both the computation of the equilibrium components of the signal, and the computation of the growth rates and rotation frequencies $\gamma(t)$ and $\omega(t)$ has to be performed over a moving time window that ends at the current sample. In both cases, a straightforward implementation yields the following formula to compute the desired result ϕ from the input Φ over the last M samples:

$$\phi[i] = \sum_{j=0}^{M-1} a_j \cdot \Phi[i-j] \tag{5.1}$$

For equilibrium subtraction, Φ are the full sensor signals. When determining the growth rate or rotation frequency, Φ are the toroidal phases or the logarithm of the quadrature amplitude (cf. Equation 4.15).

However, a direct implementation of this equation does not result in good performance, because it requires the system has to iterate through all M samples in every cycle. The GPU kernel is thus based on a different formulation of the problem.

Any set of filter coefficients a_j may be expanded in a power series as

$$a_j =: \sum_{n=0}^{N} \alpha_n j^n \tag{5.2}$$

For the least squares fitting procedures, the number of terms N in this expansion is equal to the degree p of the polynomial that is being fitted to, so N is generally much slower than M. A significant performance advantage can therefore gained by replacing the sum over M with a sum over N. This is achieved as follows. Plugging Equation 5.2 into Equation 5.1, we obtain

$$\phi[i] = \sum_{n=0}^{N} \alpha_n \sum_{j=0}^{M-1} j^n \cdot \Phi[i-j] \tag{5.3}$$

$$=: \sum_{n=0}^{N} \alpha_n K^n[i] \tag{5.4}$$

where

$$K^n[i] = \sum_{j=0}^{M-1} j^n \cdot \Phi[i-j] \tag{5.5}$$

Now, it can be shown that

$$K^0[i] = K^0[i-1] + \Phi[i] - \Phi[i-M] \tag{5.6}$$

$$K^1[i] = K^1[i-1] - M \cdot \Phi[i-M] + K^0[i-1] \tag{5.7}$$

$$K^2[i] = K^2[i-1] - M^2 \cdot \Phi[i-M] + 2 \cdot K^1[i-1] + K^0[i-1] \tag{5.8}$$

or, generally,

$$K^n[i] = \sum_{k=0}^{n} \mathscr{C}_k^n \left(K^{n-k}[i-1] - (M-1)^{n-k} \cdot \Phi[i-M] \right) + 0^n \cdot \Phi[i] \tag{5.9}$$

$$=: \sum_{k=0}^{n} \left(\mathscr{C}_k^n \cdot K^{n-k}[i-1] \right) - \alpha_n \cdot \Phi[i-M] + 0^n \cdot \Phi[i] \tag{5.10}$$

where $\mathscr{C}_k^n$ are the binomial coefficients and

$$\alpha_n = \sum_{k=0}^{n} \mathscr{C}_k^n \cdot (M-1)^{n-k} \tag{5.11}$$

This means that a second way to calculate the $\phi[i]$ is to track the K_i^n, and then calculate $\phi[i]$ from Equation 5.4. Both equilibrium subtraction and the determination of perturbation growth rate and frequency have been implemented in this way.

5.4 Thread Usage

One of the main advantages of using a GPU is the large number of available threads. The number of threads that is used to implement the adaptive control algorithm is the maximum of the number of sensors, the number of control coils, and the number of tracked perturbations. This means that at any time there are at least as many threads as there are variables that can be computed simultaneously.

Equilibrium subtraction is thus carried out for every sensor simultaneously, while determination of rotation frequency and amplitude growth is carried out simultaneously for every tracked perturbation. Similarly, control output generation (i.e., application of gain, phase, filtering, and response compensation) is done in parallel for each perturbation.

An interaction between the state variables happens in only two places: when sensor signals are converted to quadrature amplitudes and toroidal phases, and when quadrature amplitudes and phases are converted to control coil currents. In each case, the computation takes the form of a matrix application. The algorithm for such applications is simple: since there are at least as many threads available as the matrices have rows, every row of the output vector is computed by a separate thread, while threads without an output row are waiting. This method also illustrates why the use of existing GPU code libraries would not be optimal for a control system: since such libraries are optimized for very large matrices, they typically use more sophisticated techniques that involve working on individual blocks of the matrix at a time, and attempt to use multiple threads when performing the summation. While this results in a significant speedup for large matrices, for the matrices that the control system needs to handle the performance actually reduced.

5.5 Memory Layout

A GPU has several distinct types of memory with very different access times. Global memory is typically more than a GB in size and the slowest kind of memory. Accessing it takes about 400-600 clock cyles, which is about half a microsecond. However, global memory is the only memory that can be accessed externally, so input and output buffers are located in global memory.

Shared memory is typically significantly less than a megabyte, but can be accessed within 20-40 clock cycles. Once data has been read from the input buffer, it is therefore hold exclusively in shared memory. Similarly, shared memory is used to hold the $\mathbf{C}_c$, $\mathbf{C}_s$,

$\mathbf{F}_c$ and $\mathbf{F}_s$ matrices. When storing these matrices, however, special care is taken to avoid the problem of *bank conflicts*: shared memory is striped across 32 banks, which can all be accessed simultaneously. However, if the layout of the matrices in shared memory is such that elements that are required at the same time are stored in the same bank, the access needs to be serialized which incurs a performance penalty. Rather than storing the matrices as-is, they are therefore stored in row-minor order and column starts are adjusted to coincide with bank boundaries. This ensures that when multiple threads (working on different rows) are accessing the same column, each thread will access a separate bank.

As described in Section 4.6, in addition to the $\mathbf{C}_c$, $\mathbf{C}_s$, $\mathbf{F}_c$ and $\mathbf{F}_s$ matrices defining mode shape, the control algorithm has a number of additional parameters. In a CPU program, these parameters would normally be stored in regular variables. However, for GPU code this is not a good choice. This is because of the limited number of available processor registers, such variables would end up being stored in global memory, and thus take very long to access. Therefore, all remaining parameters are stored as part of instruction stream. For the GPU, they appear as hardcoded values, but to the programmer they appear as constants whose values are defined by the C header file generated by the preprocessor.

The last type of performance critical data that has to be stored is the sequence of full sensor signals. While the continuous fitting procedure described in Section 5.3 reduces the number of coefficients from the number of samples in the time window to the degree of the polynomial, it is still necessary to store the entire sequence of measurements because any update requires access to both the first and last measurement in the considered time window. The resulting amount of data is too large for shared memory and thus stored in global memory. However, the resulting performance drag can in this case be eliminated almost completely using data pre-fetching, i.e. by requesting the data from global memory before its actually needed. Since only individual cache lines can be prefetched, this requires that the sequence of measurements is stored with the sensor index varying most quickly (i.e., successive samples are from different sensors but measured at the same time, rather than from different times but measured by the same sensor).

Chapter 6

Application to the HBT-EP Tokamak

This chapter describes the setup and configuration of the adaptive control algorithm introduced in Chapter 4 for use on the HBT-EP tokamak.

Generally, parameters have been chosen to maximize effects on the plasma, to allow comparison of the results with earlier experiments, and to minimize the number of potential error sources.

The system is set up to use HBT-EP's large control coil set to maximize the achievable magnetic field strength. Poloidal sensors are used as control inputs to reduce the effects of eddy currents and minimize direct coupling to the control coils. Control coils and sensors are each arranged in a 4x10 grid. For increased tolerance to a changing major radius, the expected helical perturbation is tracked in form of four separate $n = 1$ modes (one in each toroidal array) that are independent in amplitude and phase, but coupled in frequency. Equilibrium subtraction is done using 1st degree polynomials over a period of 600 microseconds. Feedback latency at 4 µs is found to be 16 µs. Amplifier and eddy current response is measured and fitted to 3rd degree polynomials for phase and gain.

6.1 HBT-EP Plasmas

The HBT-EP tokamak has pioneered active feedback control of magnetic perturbations [11], and later experiments have continuously advanced the complexity of the control systems and algorithms [38, 32, 34]. HBT-EP's extensive magnetic diagnostics and control coils [56, 40, 44] make it very attractive for feedback control experiments.

Figure 6.1 shows the typical evolution of an HBT-EP plasma. After the toroidal field is up, the plasma is generated by using the ohmic heating coil as a transformer to accelerate seed electrons in the toroidal direction. Those electrons create further free electrons and

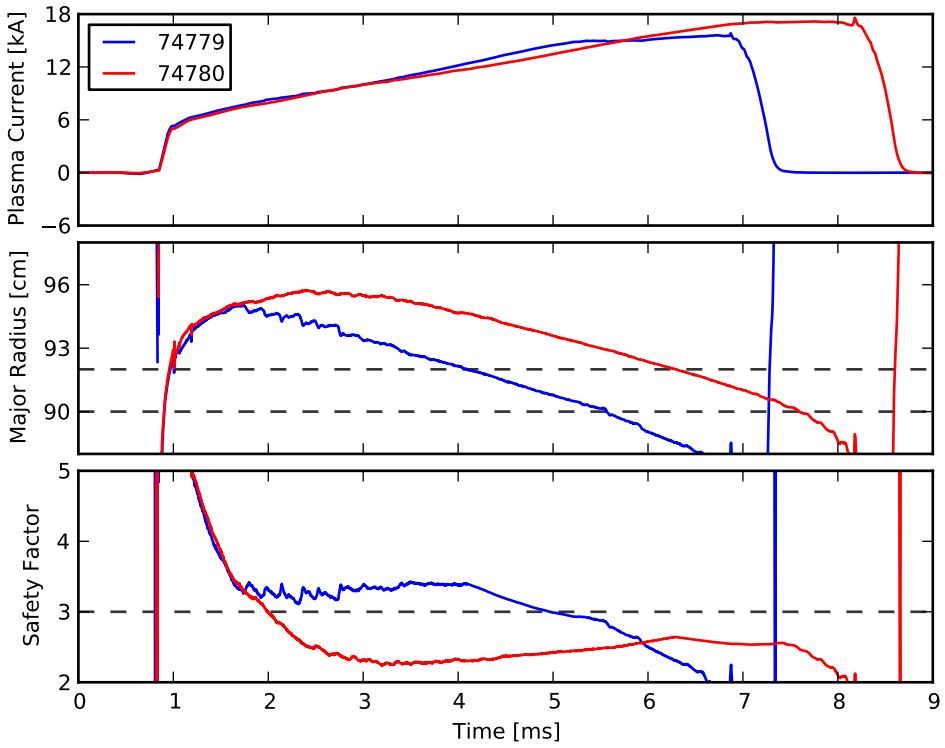

Figure 6.1: Evolution of plasma parameters in two consecutive HBT-EP shots. Even though no controlled parameters have been changed, the shot evolution differs considerably. In the first shot, a tearing mode arises briefly during start-up (2-3 ms), but changes the plasma evolution throughout the rest of the shot.

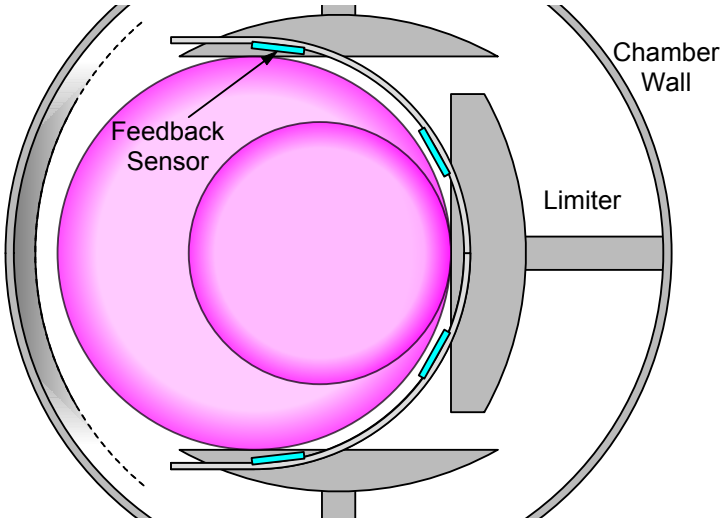

Figure 6.2: Location of feedback sensors relative to the plasma surface. The control coils are at the same poloidal locations as the sensors but toroidally behind. For plasma major radii above 92 cm, the plasma surface recedes from the lower and upper sensors. (Figure courtesy of Jeffrey Levesque.)

ions by impact ionization of deuterium molecules. Once an initial toroidal plasma current is established, the vertical field is ramped up to create the radial force necessary for a stable equilibrium with toroidal flux surfaces. This initial breakdown phase takes between 250 µs and 600 µs. After the breakdown, additional capacitor banks for vertical field and ohmic heating coils are switched on to inductively drive the plasma current until the end of the shot. A more detailed description of HBT-EP operation can be found in Gates [24].

For HBT-EP, the major radius is defined as the current centroid as measured by a cosine rogowski. The edge safety factor is calculated from major radius, minor radius and plasma current using a cylindrical approximation. Typical HBT-EP plasmas slowly fall from larger to smaller major radii.

Due to the limiter positions, the plasma achieves a constant, maximal minor radius of 15 cm only for major radii between 90 and 92 cm. As illustrated in Figure 6.2, HBT-EP has both inboard, outbord, upper and lower limiters. The upper and lower limiters limit the maximum minor radius of the plasma to 15 cm. However, this minor radius can only be achieved when the plasma center (major radius) is between 90 and 92 cm. For major radii above 92 cm, the plasma becomes outboard limited, and the minor radius decreases linearly with increasing major radius. For major radii below 90 cm, the plasma becomes inboard limited, and the minor radius decreases linearly with decreasing major radius. Typical HBT-EP plasmas start

out outboard limited with a small minor radius, then grow to the maximum of 15 cm, and typically disrupt around the time that the major radius hits 90 cm. This is reflected in the edge safety factor profile of the second shot in Figure 6.1, which starts to rise at 3 ms due because rising minor radius dominates over the effects of the increasing plasma current, reaches a peak at 6.3 ms and then falls again when the minor radius becomes constant and plasma continues to rise.

HBT-EP does not have facilities to dynamically adjust global plasma parameters during a shot. Instead, different kinds of shots are achieved by changing the voltage levels and triggering times of the different capacitor banks. For this reason, the evolution of consecutive HBT-EP shots often differs significantly even if no adjustable parameters have been changed. For example, in the shots plotted in Figure 6.1, all adjustable parameters are identical, yet the plasma evolution differs significantly (in this case due to the excitation of a fast tearing mode in the first shot around 2 ms).

6.2 Sensors and Actuators

Of the 216 magnetic sensors installed on HBT-EP, 80 "feedback sensors" (shown green in Figure 1.3) are available for real-time control. These feedback sensors measure poloidal and radial field and are placed in a 4x10 grid on the shells surrounding the plasma, covering 360 degrees of toroidal angle and 240 degrees of poloidal angle. In order to minimize effects of shell eddy currents and direct coupling between control coils and sensors, only the 40 poloidal measurements have been selected for feedback experiments. All magnetic sensors also measure the time derivative of the magnetic field rather than the field itself and are therefore integrated prior to all other control processing.

In order to control the plasma, HBT-EP has a total of 120 control coils. These coils are split into 3 sets, that are each arranged in the same grid as the feedback sensors (cf. Figure 1.3). The coils are driven by Crown XLS5000 audio amplifiers. Formally, these amplifiers accept input amplitudes up to 2 V. However, at high enough frequencies good results have been obtained with amplitudes up to 6 V. To maximize the effect of the control system, the large control coil set was selected as the actuator for feedback experiments. These coils produce peak normal magnetic fields of 1.5 G per Ampere of coil current at a minor radius of 15 cm. For feedback experiments, each of the toroidal control coil arrays was set up to produce the same $n = 1$ configuration that is measured by the corresponding sensor array.

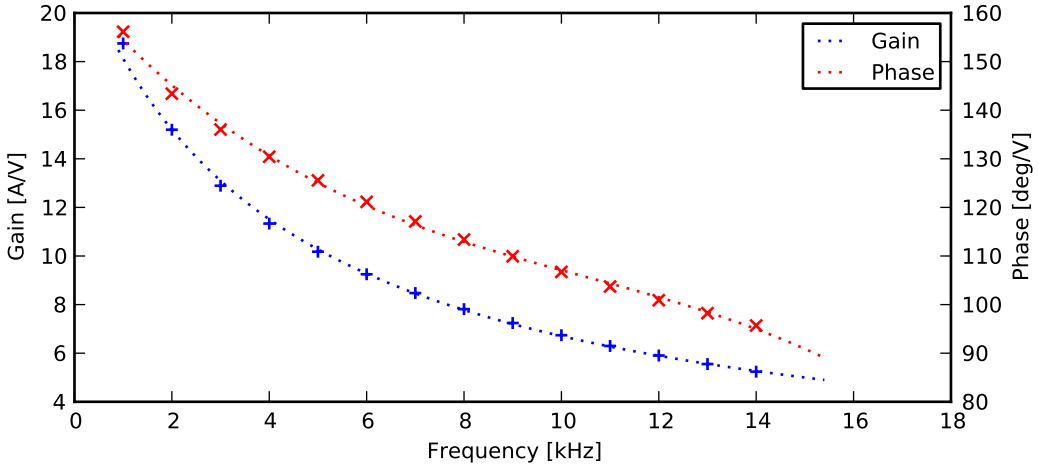

Figure 6.3: Gain (in Ampere per Volt) and phase response of the control coil and amplifier system and polynomial fits.

Since the control coils are located on top of the conducting wall segments, they induce strong eddy currents that momentarily oppose any action of the control coils. Similarly, the audio amplifiers have frequency dependent gain and phase response. Both of these factors have to be taken into account when generating the control output.

The response of coils, eddy currents and audio amplifiers has been measured on the bench by feeding sinusoidal test signals of different frequencies into the amplifiers, and measuring both the resulting control coil currents and magnetic fields underneath the wall segment holding the coil. The currents were measured using a shunt resistor, and the magnetic fields with a hall probe. The results are shown in Figure 6.3 and Figure 6.4. As expected, the gain in both currents and fields decreases with increasing frequency, and the decrease is stronger in the fields due to the additional eddy current effects. When looking at the phase response, the phase in the control coil currents changes by 90° over the tested frequency range. This is consistent with the response changing from being mostly due to control coil resistance to being dominated by control coil inductance. The change in the phase of the generated fields changes by a total of 120 degrees, which again indicates the significant eddy current effects. Both gain and phase measurements have been fit to a 2nd and 3rd order polynomials respectively so that the coefficients can be used in the control system to determine the necessary compensation for any frequency. (In the figures, the gain may appear to be better described by an exponential, but this is not actually the case.)

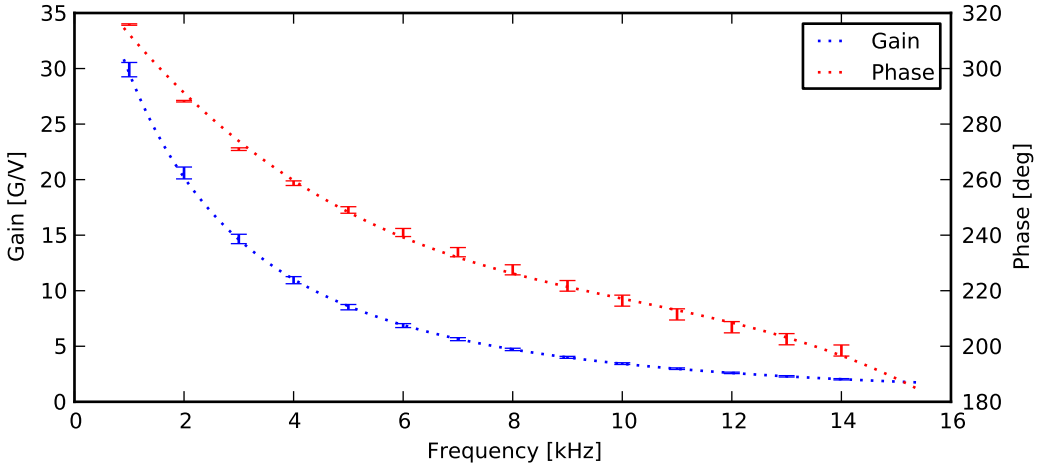

Figure 6.4: Gain (in Gauss per Volt) and phase response of the combined control coil, shell segment eddy current and amplifier system and polynomial approximations. The frequency dependent changes in gain and phase have to be taken into account when generating control output signals.

6.3 Feedback Parameters

Since neither control system nor control algorithm can be tested separately, feedback experiments will necessarily test both of them simultaneously. For this reason, the parameters of the feedback system have been chosen to

1. maximize the effect on the plasma to increase the likelihood of an observable response
2. resemble previous feedback experiments (in which a response was observed) as much as possible
3. use only the minimum functionality of the feedback algorithm that is expected to give results and reserve more advanced features for later experiments

The length of the fitting window and degree of the polynomial used to determine the equilibrium components in real-time was chosen by scanning the parameter space and selecting the configuration that maximized agreement with the results from biorthogonal decomposition. An example of such a scan is shown in Figure 6.5 and indicates an optimal window length of 600 µs with a linear polynomial. These results vary little between different shots and have been used for almost all feedback experiments.

In order to account for the changing poloidal spectrum of the perturbations, the control algorithm was set up to control four independent perturbations with toroidal mode number

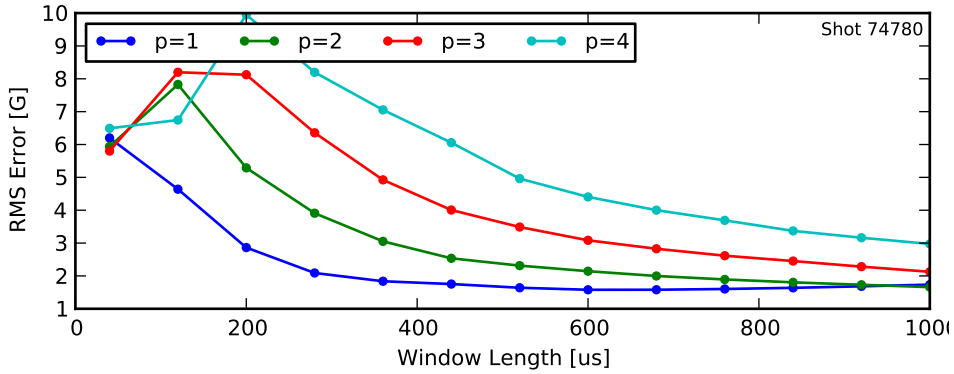

Figure 6.5: Deviation between real-time calculation of equilibrium components in sensor signals and post-shot biorthogonal decomposition for different fitting window lengths and polynomial degrees p. Errors are averaged over time and sensors. The position of the minimum varies very little between different shots.

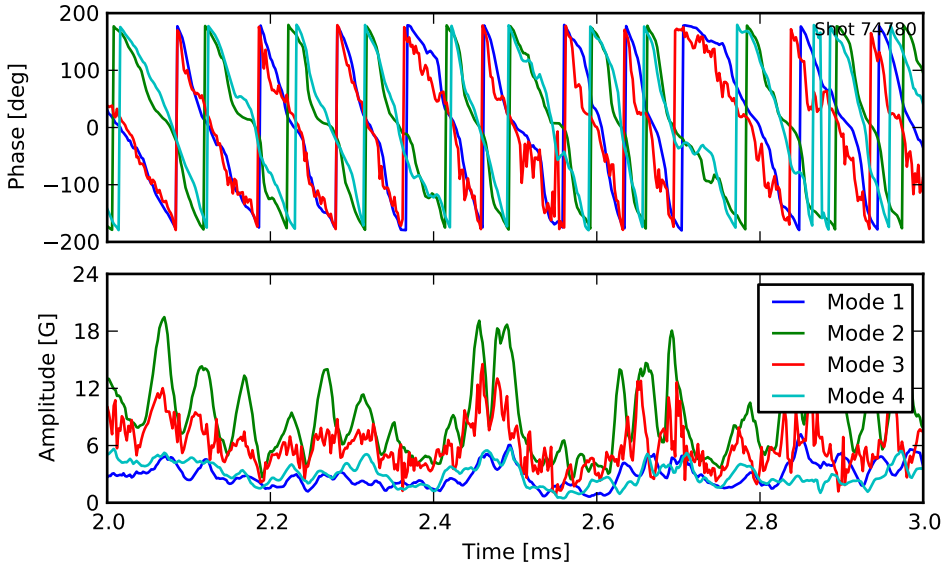

Figure 6.6: Phase and amplitude evolution of the $n = 1$ mode in the four toroidal sensor arrays. The individual modes are offset in phase and the scale of the amplitudes differs, but the temporal evolution is similar and indicates that the modes are really components of the same rigid, rotating structure.

$n = 1$, corresponding to the four toroidal feedback sensor arrays (shown green in Figure 1.3). An example of phase and amplitude evolution of these modes is plotted in Figure 6.6. As expected, the time evolution of all four modes is similar, but individual modes are offset from each other in phase and differ in the overall scale of the amplitude. This is consistent with the assumption that the individual modes are part of the same rigid, rotating perturbation. In the control algorithm, this assumption is reflected in a coupling of the rotation frequencies, i.e. the algorithm performs the least squares fit over the phase evolution of all four modes to the same rotation frequency. Based on the phase variations that have been observed in shots without feedback control, a time window of 400 µs has been chosen for determination of the rotation frequency, and the model-based filter coefficient α (cf. Equation 8.6) was set to one (corresponding to the least amount of smoothing).

Tracking the mode growth rate posed a more complicated problem, as in HBT-EP shots the quadrature amplitude is not growing or decaying exponentially but fluctuating around a saturated amplitude with periods of less than 100 µs. Growth-rate-based compensation and filtering was therefore disabled.

In order to get a control response as smooth as possible, the control system was ran with the minimum cycle time of 4 µs. The resulting latency (including the time required for computations) was measured as described in Section 2.6 and found to be 16 µs. Figure 6.7 summarizes the implementation and parameters of the adaptive feedback algorithm as implemented for HBT-EP. With the chosen parameters, the algorithm can be split into two parts: a traditional feedback loop assuming a fixed system model (as it was used in previous HBT-EP experiments), and a second loop that dynamically adjusts the system model used in the first loop.

6.4 Latency and Response Compensation Testing

The last test performed before beginning actual feedback experiments was to verify the latency and coil/amplifier response compensation over the entire open-loop control chain. To that end, a function generator was used to generate artificial sensor signals corresponding to a perfect $n = 1$ mode rotating at fixed frequencies. In addition to being used by the control system, all feedback sensors are also digitized in HBT-EP's data acquisition system. This system also digitizes the currents in every control coil, which are measured over shunt resistors. For purposes of this test, a normally otherwise used digitizer was used to record the

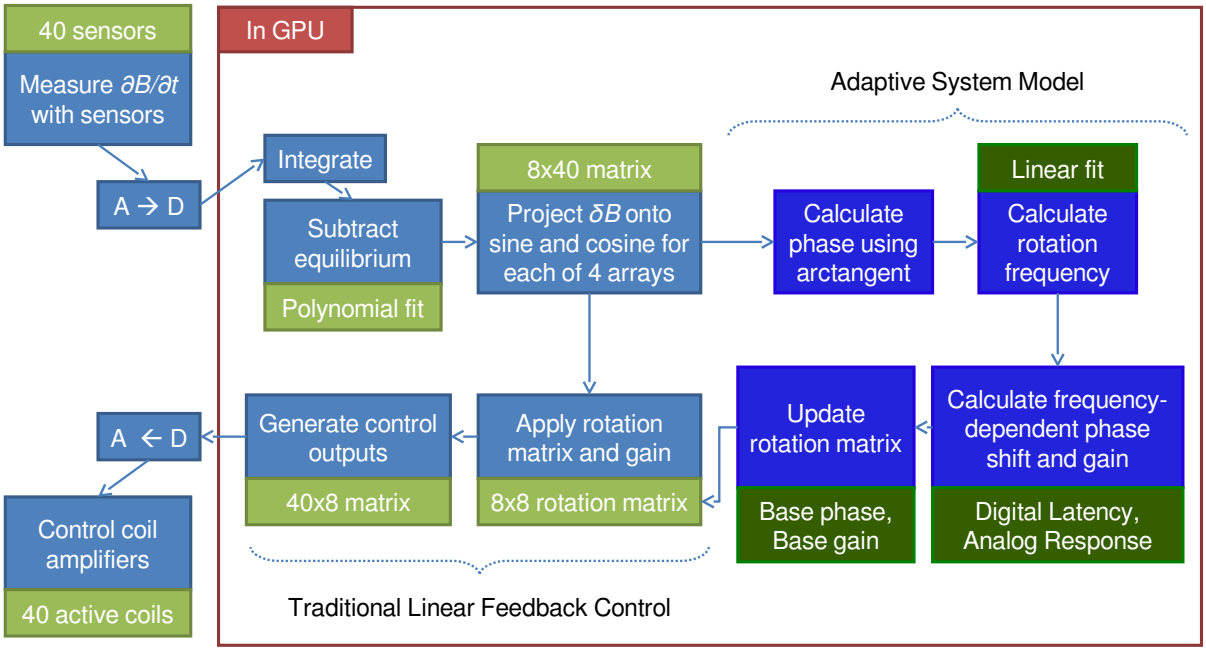

Figure 6.7: Block diagram of the adaptive control algorithm as implemented for the rotating perturbations occuring in the HBT-EP tokamak. The algorithm can be split into two parts: a traditional feedback loop that resembles earlier experiments, and a new loop implementing the adaptive nature that updates the model used by the first loop.

output of the control system before it entered the amplifiers.

To verify the correct latency compensation, the latency assumed by the control system was temporarily set to zero (resulting in no corrections). When comparing the toroidal phase computed from the artificial sensor measurements with the phase computed from the control output prior to amplification, the control output was found to consistently lag behind by 16 µs. As expected, the lag did not change with rotation frequency, which is consistent with it reflecting the time required for A-D/D-A conversion, transfer and control computations. The control algorithm was then set back to the intended configuration that assumes the correct latency of 16 µs. As described in Section 4.4, the control algorithm then uses the (fixed) 16 µs lag and the (dynamically determined) rotation frequency to calculate the resulting the phase shift and applies this shift to the control output. With active latency compensation, no phase differences were found between sensor measurements and control system output.

With this stage of the control cycle verified, the same test was performed while measuring the phase difference between sensors and control coil currents. Without any coil/amplifier response compensation, the phase difference agreed with Figure 6.3 as expected. If the control system was set up to compensate for the response using the polynomial fitting coefficients, no phase differences were found. With the control system set up in its final configuration, where it compensates for the response shown in Figure 6.4, a frequency dependent phase difference occurs. This is desired, because the system is expected to eliminate any phase differences between the measured mode and the applied magnetic fields, while the test setup measures the difference between sensors and control coil currents (and therefore does not capture the effects of eddy currents in the shell segments).

Chapter 7

Experimental Results from HBT-EP

This chapter describes the results of exciting and suppressing magnetic perturbations on the HBT-EP tokamak using the adaptive control algorithm (cf. Chapter 4 and 6) running on a GPU-based control system (cf. Chapter 2). Feedback effects are quantified by their effects on the frequency spectrum of perturbations and averaged over multiple shots to compensate for shot-to-shot variation. It is found that the amplitude of the dominant -8 kHz mode can be feedback suppressed down to 40% of the uncontrolled amplitude. Depending on feedback gain, the system may excite a slowly rotating -1.4 kHz mode at the same time. Mode amplification is observed by up to a factor of two. The time window where amplification can be achieved is found to differ from the time window where the mode can be suppressed. Feedback phases between suppression and amplification result in a speed up or slow down of the mode rotation frequency. The performance of the control system is found to match and exceed previous results without requiring any tuning of model or feedback parameters.

7.1 Analysis Techniques

7.1.1 Shot Selection

The natural variability of HBT-EP shots make direct comparisons between individual shots with- and without active feedback (or with different feedback parameters) very hard to interpret. Therefore, the majority of the results presented in this chapter have been obtained using statistical methods. This was done by taking ensembles of shots, with the feedback parameters varying between ensembles but not between individual shots of an ensemble. An automated selection algorithm was then run over the entire corpus (consisting of the union of all ensembles) to get a subset of shots with similar global parameters. After this

elimination, the shots were repartitioned into ensembles with different parameters which were then compared.

By running the selection algorithm on the corpus rather than on individual ensembles, any changes in global plasma evolution caused by different feedback settings will be actively suppressed and not show up in the analysis. However, when running the selection algorithm separately for each ensemble, there is a risk of introducing artificial differences, because individual ensembles may converge on different global plasma evolutions. When weighting these alternatives, the first approach was chosen as the more appropriate one. Even though this means that some effects may not show up in the analysis, it increases the confidence in the significance of any results that do show up.

The selection algorithm works by first identifying the largest window of smooth plasma evolution in every shot, where a smooth evolution is defined by restrictions on the rate of change and value of plasma current, major radius and edge safety factor “q”. For all further processing, attention is restricted to this window. The next step is to align all shots by the edge safety factor peak that occurs when the major radius hits 92 cm (for an explanation for this peak, see cf. Section 6.1). Having applied the different time shifts, the algorithm iteratively computes the average edge safety factor evolution and then drops the shot with the largest deviation from the average. This procedure is repeated until the standard deviation (averaged over time) of the remaining shots falls under 0.05.

Figures 7.1 and 7.2 illustrate the automatic selection procedure. The first figure shows the edge safety factor evolution of all shots that were taken for the first set of experiments. All shots have been time shifted such that their peak q values are reached at 5 ms. Since only regions of smooth plasma evolution are considered, the number of plotted shots changes over time and is indicated on the right axis. Figure 7.2 shows only the shots that have been selected for analysis. Evidently, the standard deviation is significantly lower, but the total number of shots has dropped from about 60 to 33.

7.1.2 Campaigns

In principle, every shot taken for this thesis could be considered part of the same corpus. However, it turned out that a better approach is to introduce the notion of campaigns. In each campaign, a specific aspect of the control system was investigated. Every campaign thus formed its own corpus with its own set of ensembles. Since the shots of a campaign were mostly taken in one continuous sequence, this minimized the effects of day-to-day variations

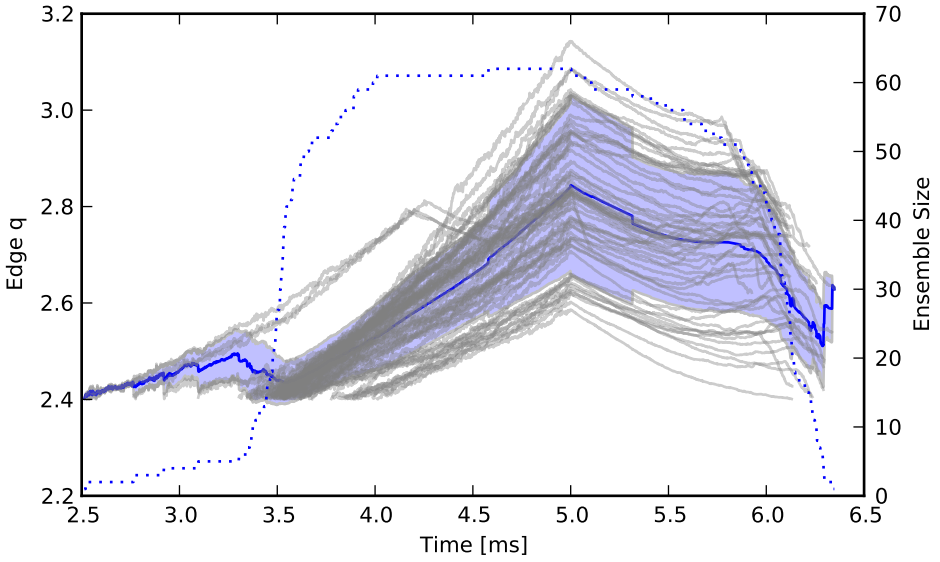

Figure 7.1: Edge safety factor (q) evolution of all shots taken for the first experimental campaign. The blue line is the average, and the shaded region indicates one standard deviation. The dotted line indicates the number of shots that are contributing to the average at a given time.

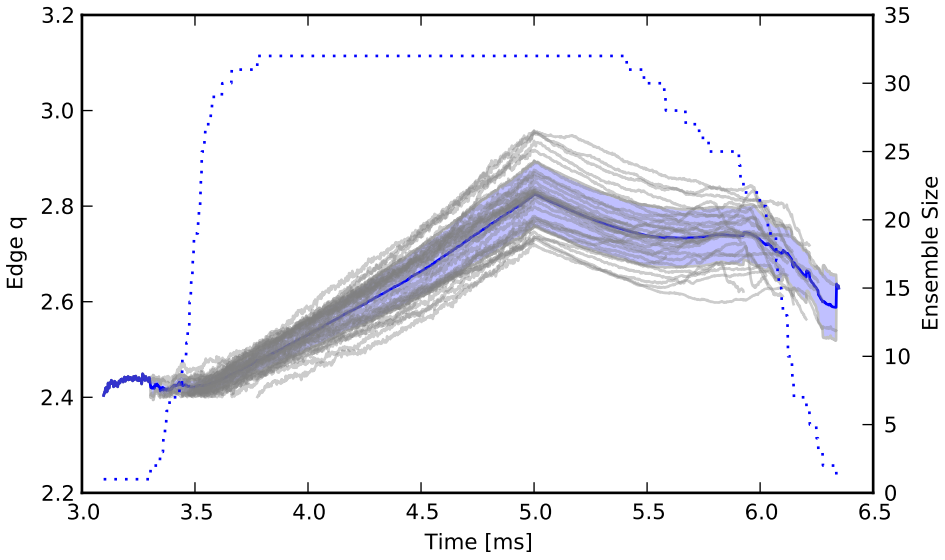

Figure 7.2: Edge safety factor (q) evolution of the shot subset from Figure 7.1 that was selected for further analysis by the automated selection algorithm.

and allowed a larger fraction of shots to be retained for analysis. A total of 5 campaigns was conducted for this thesis:

1. A phase scan in 90° steps consisting of 66 shots, 33 of which were selected for analysis.
2. A phase scan in 15° steps in the vicinity of 100° and 280°, with 40 out of 67 shots selected for analysis.
3. A gain scan at 85° phase, with 35 out of 90 shots selected for analysis.
4. A phase scan in 15° steps in the vicinity of 100° with slowed plasmas. 26 out of 38 shots were selected for analysis.
5. A phase scan in 90° steps with slowed plasmas. 38 of 46 shots were selected for analysis.

Each campaign also included its own control ensemble of shots without feedback.

7.1.3 Frequency Spectra

Most effects of the feedback control system are best observed in frequency space. As explained in Section 6.2, the control algorithm has been set up to track four $n = 1$ modes. The (complex) spectral amplitude $f(\omega)$ of a mode that has measured (real) amplitudes $A_c(t)$ and $A_s(t)$ for its cosine and sine components respectively is defined as

$$f(\omega) = \mathscr{F}\left[A_c(t) + i\,A_s(t)\right] \tag{7.1}$$

where $\mathscr{F}$ denotes the fourier transform. With this definition, $|f(\omega)|$ gives the amplitude of the mode at frequency ω, and $\arg(f(\omega))$ gives a phase offset. The phases offset is generally not important for the purposes of this chapter, and frequency plots thus show just $|f(\omega)|$.

It should be noted that $f(\omega) \neq f(-\omega)^*$ (with the asterisk indicating complex conjugation), because the input to the fourier transform is complex. For an idealized mode that rotates at a fixed frequency ω_0, $f(\omega) = \delta(\omega \pm \omega_0)$, and the sign distinguishes between the directions of rotation.

As mentioned before, most of the analysis presented in this chapter is based on averaging over multiple shots. When computing averages for frequency spectra, however, the averaging is additionally performed over the individual tracked modes: since it is assumed that every mode is just a projection of the same rotating helical perturbation, their individual spectra

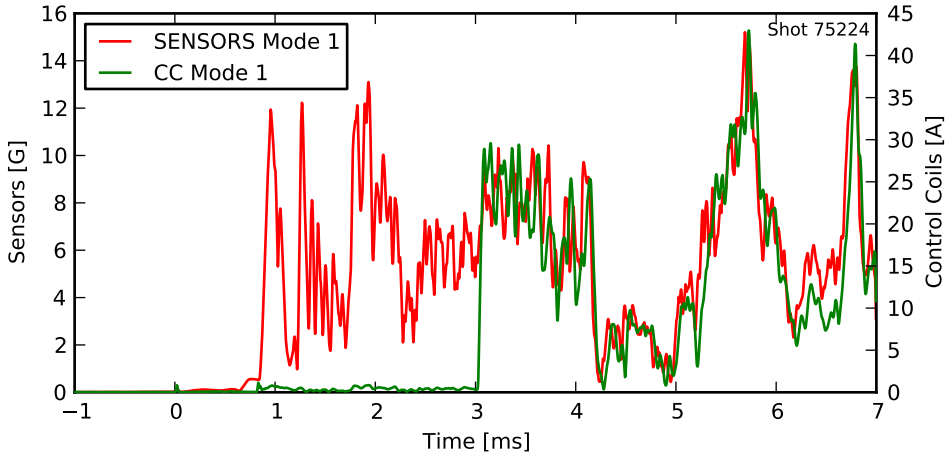

Figure 7.3: Mode amplitude in control coils and sensors. The feedback system is activated at 3 ms. Prior to activation, the plasma already induces small currents in the control coils.

should converge. When talking about frequency spectra, averages over N shots thus involve $4N$ terms, and statements about the average amplitude should be understood as statements about the quadrature sum of the individual modes. In plots of averaged quantities (e.g. in Figure 7.5), shaded areas indicate one standard deviation.

7.2 Phase Scan

The first experimental campaign was aimed at getting an overview of the effects of the base feedback phase and identifying the ranges of positive and negative feedback. The feedback system was turned on at 3 ms and left running until the end of the shot. An example of the typical evolution of the modes measured in the control coil currents and magnetic sensors is plotted in Figures 7.3 and 7.4. The activation of the feedback system can clearly be seen at 3 ms and results in a jump of both amplitude and phase difference, but even before the feedback system is started, the plasma induces weak currents in the control coils. The "natural" phase difference of 100° at this time roughly indicates the phase offset for which positive feedback effects are expected. The agreement is not expected to be perfect, because strongest feedback effects occur if the magnetic field from the perturbations is in phase with the magnetic field due to the control coil currents, rather than in phase with the currents.

The analysis of the feedback effects was performed with the same set of sensors that are

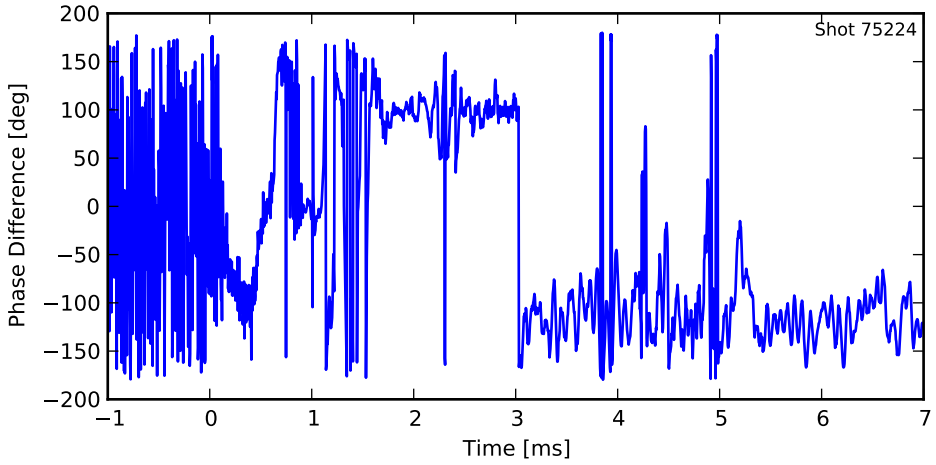

Figure 7.4: Phase difference between the mode measured in control coil currents and magnetic sensors. The feedback system is activated at 3 ms. The phase difference prior to 3 ms indicates the phase region where positive feedback effects are to be expected.

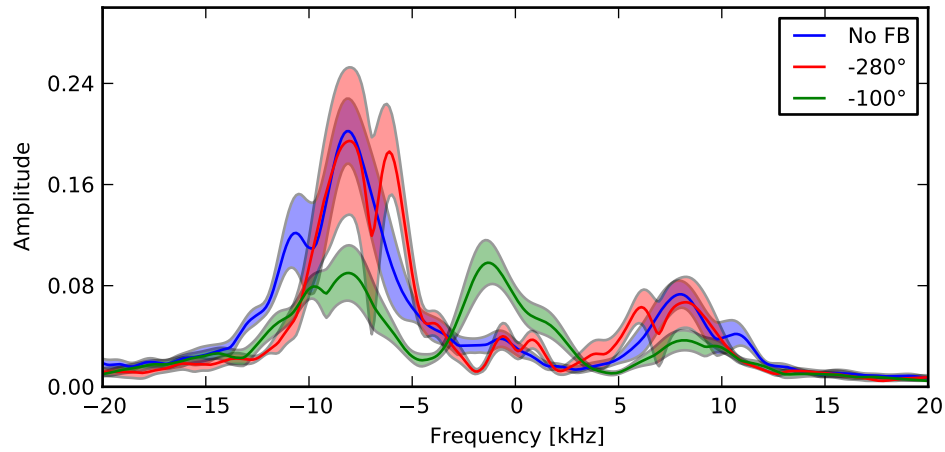

Figure 7.5: Frequency spectrum of magnetic perturbations without feedback, and with feedback with -100° and -280° phase. At -100°, the amplitude of the -8 kHz mode is suppressed to 40% of its no-feedback value, but an additional -1.4 kHz mode is excited. At -280°, no significant amplification is observed, but shot-to-shot variability increases by a factor of two and a new mode is excited at -6 kHz.

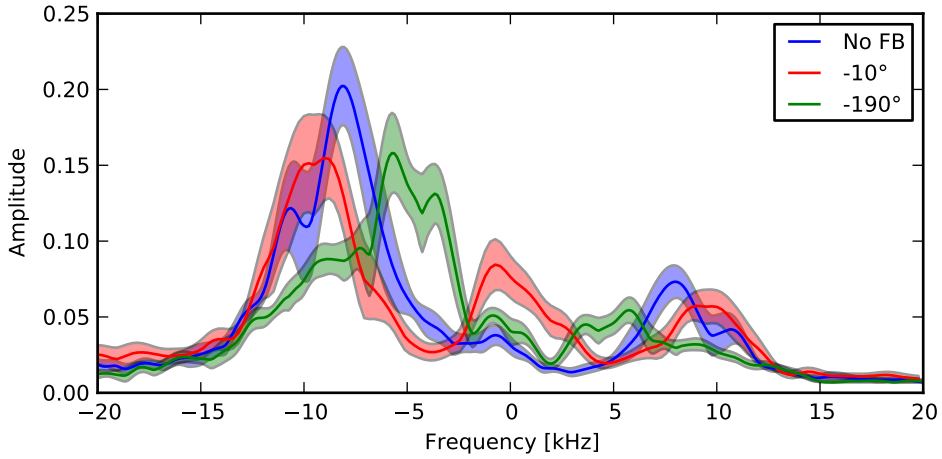

Figure 7.6: Effects of -10° and -190° phased feedback compared to the no-feedback case. Both phase angles suppress the dominant -8 kHz mode, but suppression is not as strong as for the -100° feedback shown in Figure 7.5. However, -10° and -190° phasing results in an additional speed up and slow down of the mode rotation frequency respectively.

used for real-time control and restricted to the time window in which the plasma had the maximal minor radius of 15 cm. The results are summarized in Figures 7.5 and 7.6. As one can see, without feedback the spectrum is dominated by a peak at -8 kHz, corresponding to a rotating mode.

In agreement with the passive measurements, negative feedback effects were most strongly observed at a phase of -100°, where the amplitude of the -8 kHz mode was reduced from 0.2 to 0.08, i.e. by about 60%. In addition to that, however, a strong -1.4 kHz mode has been excited to about 0.09. This mode only appears in the presence of a plasma, i.e. it is not a self-oscillation of the control system.

Looking at the -280° phasing where one would correspondingly expect positive feedback effects, the fluctuations between different shots have increased by a factor of two, but the mean amplitude of the dominant mode is unchanged compared to the no-feedback case. However, compared to feedback at -100°, the frequency of the slow mode has increased from -1.4 to -6 kHz.

At intermediate phases, the control system either speeds up (-10°) or slows down (-190°) the dominant mode in addition to a roughly 25% suppression compared to the no-feedback case. Relative to feedback with the suppressing phase of -100°, the slow mode is further amplified and sped-up at -190°, but suppressed and slowed-down at -10°.

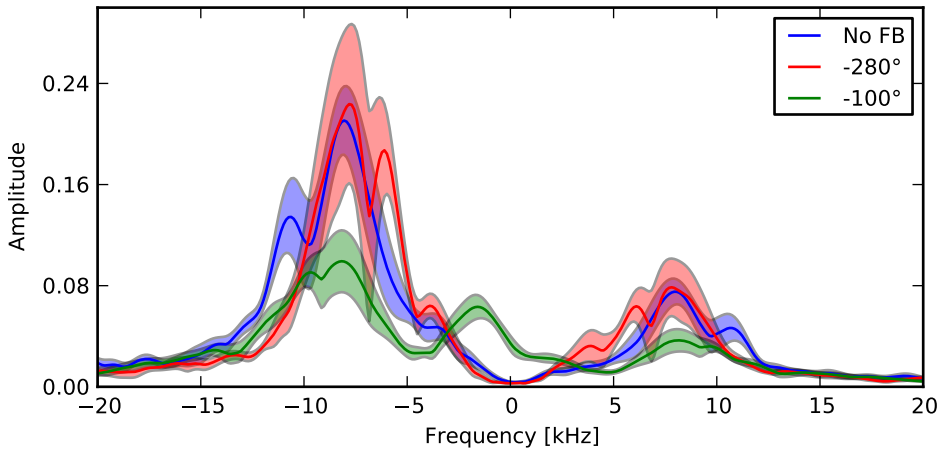

Figure 7.7: This plot shows the same data as Figure 7.5. However, in this case the equilibrium signal components were subtracted with the real-time algorithm used by the control system. Note how the amplitude of at slow frequencies, and especially of the -1.4 kHz mode, is significantly reduced.

It is interesting to consider the effects of the real-time equilibrium subtraction method used by the control system when compared to the BD-based subtraction used in the analysis. Figure 7.7 shows the same data as Figure 7.5, but this time the equilibrium was subtracted using the real-time algorithm. It can be seen that for the control system, the -1.4 kHz mode appears to have a much lower amplitude than it actually has. The existence of a second mode with the same spatial structure but a different rotation frequency can safely be assumed to be a limiting factor in the achievable suppression rates, as the control algorithm will struggle to fit the resulting signals to a single frequency.

7.3 Gain Scan

The phase scan described above was performed with an essentially arbitrary gain that was chosen such as to use about 50% of the available control power. The next analysis therefore concentrated on the effects of feedback gain. Shots were taken with varying gain and constant, suppressing phase. The analysis was performed as for the phase scan, and the results are highlighted in Figure 7.8. This figure shows the frequency spectrum of the lowest tested gain, the highest gain that could be applied without causing a plasma disruption, and the no-feedback case. In terms of RMS coil currents, the lowest gain corresponds to about 0.3

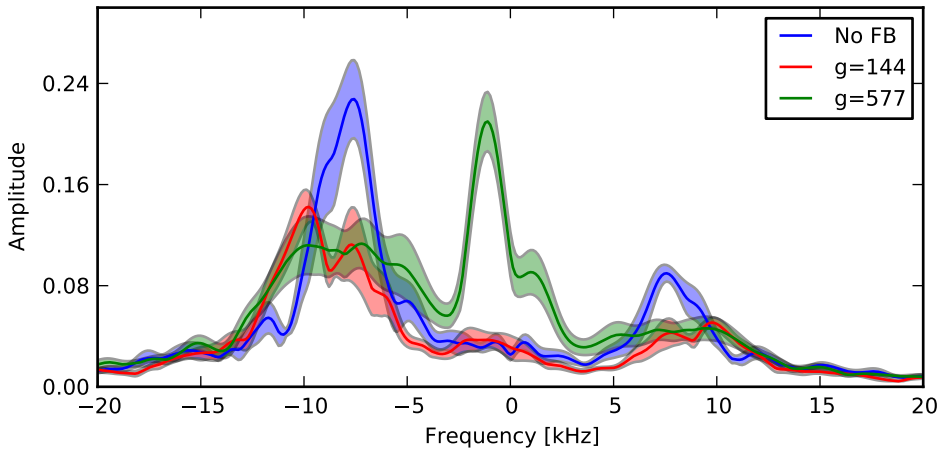

Figure 7.8: Mode spectrum for different feedback gains. Even a quadrupled frequency does not significantly change suppression at 8 kHz. However, the larger gain is responsible for the low frequency peak which is known to negatively affect mode suppression.

Ampere. Evidently, the choice of feedback gain in this range has very little effect on the effectiveness with which the -8 kHz mode can be suppressed. However, a higher gain results in the excitation of an additional mode that rotates slowly with about 1.4 kHz. This mode has already been observed in the phase scan and its appearance reduces the effectiveness of the control algorithm, because it invalidates the assumption of rigid rotation at constant frequency. It is possible that this reduced efficiency is balanced by the increased coil currents at increased gain, resulting in the apparently invariant suppression efficiency.

A second effect is that lower feedback gains seems to result not just in mode suppression, but also in a slight speed up of the rotation frequency. Considering that the same effect has been observed for non-ideal phases in the phase scan, it is likely that a slightly different phase at lowest gain would achieve even better suppression without affecting rotation frequency at all.

7.4 Major Radius Dependence

An important feature of HBT-EP plasmas is the relationship between plasma major radius and plasma minor radius. As explained in Section 6.1, the plasma achieves a constant, maximal minor radius of 15 cm only for major radii between 90 and 92 cm when the plasma is up/down

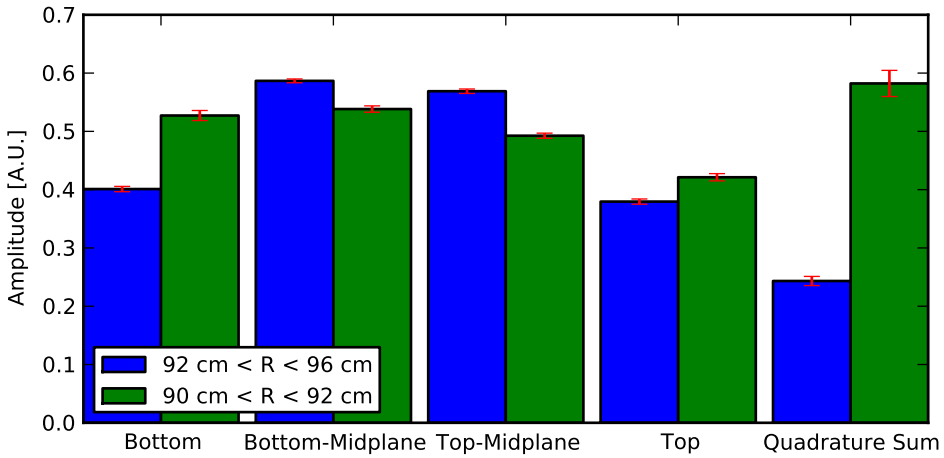

Figure 7.9: Comparison of average mode amplitudes in the different sensor arrays before and after the plasma has reached its maximum minor radius. Blue and green bars have been normalized to their respective quadrature sum. Overall mode activity at maximum minor radius is more than twice the other value, and significant amplitude differences are seen in the sensor arrays.

limited. This effect is important, because when the major radius is above 92 cm and the plasma thus outboard limited, the outboard coils and sensor arrays are much closer to the plasma surface than the sensors located on the top and bottom of the shell (cf. Figure 6.2).

Figure 7.9 compares the average mode amplitude measured by the different sensor arrays before and after the major radius has reached 92 cm. The rightmost two bars indicate the quadrature sum of the arrays in each case, and the individual arrays have been normalized to this value. From the quadrature sum, one can thus see that the average mode amplitude for $R < 92$ cm is more than twice as high as for $R > 92$ cm. From the relative heights of the individual sensors, it can be seen that for $R > 92$ cm the midplane sensors measure a significantly larger amplitude than the top and bottom arrays. This effect is expected, as the smaller minor radius reduces the coupling of the top and bottom arrays. For $R < 92$ cm, the variation between the arrays is smaller, and bottom and midplane-bottom sensors measure higher amplitudes than top and midplane top sensors. A possible explanation for this discrepancy is a vertical displacement of the plasma.

The different couplings of the sensor arrays motivate a look at the feedback effects in the time window with $R > 92$ cm. Figure 7.10 shows the frequency spectra for the same shot ensembles used in Figure 7.5, but calculated over the entire life time of the plasma after the activation of the feedback system (rather than the window in which $R < 92$ cm). In

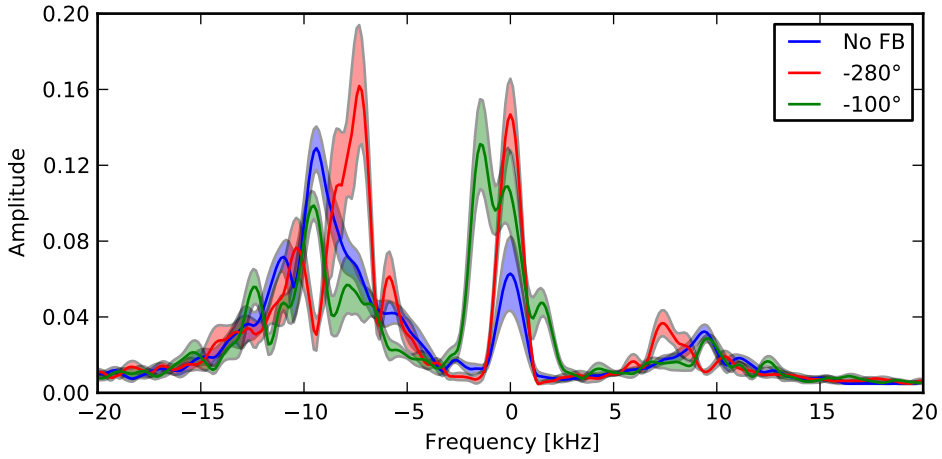

Figure 7.10: Frequency spectrum of poloidal perturbations, calculated over the entire window for which feedback is active. In contrast to Figure 7.5,this also includes the initial period where the plasma minor radius is less than 15 cm and the plasma is not equidistant from all feedback arrays.

this case, suppression of the dominant 8 kHz mode at -100° phasing becomes much less efficient, and the amplitude of the excited 1.4 kHz mode almost quadruples. Even more pronounced are the effects on the positive feedback phase at -280°. While Figure 7.5 showed no significant amplification at all, the amplitude of the -8 kHz mode is now increased by a factor of two. In addition to that, there is now also a zero frequency perturbation with an amplitude comparable to the -8 kHz mode in all three cases.

While there is not enough data to deduce the precise factors responsible for each of the observed effects, there are possible explanations for most of them. Firstly, it can be assumed that the reduced coupling at smaller minor radius will reduce the accuracy with which modes can be detected and controlled, resulting in a reduced suppression accuracy. The significantly enhanced mode amplification at positive feedback phase may be caused by an increased sensitivity of the plasma due to the different edge safety factor, but may also mean that the mode reaches its saturated amplitude at roughly the same time when the plasma reaches its full minor radius. In this case, the control system is able to amplify the mode to the saturated amplitude in the time period before that, while afterwards the mode has already saturated and the amplitude can only be changed very little.

The emergence of the zero frequency perturbation is more puzzling. It should be noted that this perturbation is invisible for the control system (as the real-time filtering algorithm will eliminate static perturbations), yet is amplified to the same extent as the -8 kHz mode.

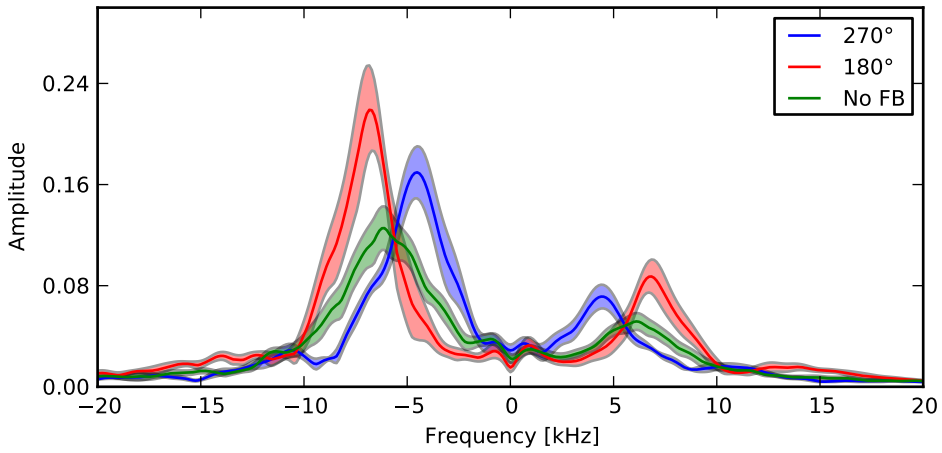

Figure 7.11: When plasma rotation is slowed with a biased probe, no negative feedback is observed at any phase. However, mode amplification factor and rotation frequency is found to vary with the applied phase. Furthermore, even high feedback gains do not result in excitation of the slow 1.4 kHz modes that have been observed in naturally rotating plasmas.

One therefore has to conclude that the static perturbation is produced by the plasma in response to rotating applied field.

7.5 Slowed Plasmas

For the last set of experiments, the plasma rotation was slowed using a biased probe that introduces a radial current. This radial current then results in a toroidal Lorentz force that slows down the plasma rotation. For these experiments, the window length for real-time equilibrium subtraction had to be doubled to 1.2 ms to maximize agreement with biorthogonal decomposition. The mere presence of the bias probe also significantly changes the plasma evolution. When looking at the edge safety factor evolution (not plotted), insertion of the bias probe changes the peak edge safety factor from 2.8 to 3.0. Mode amplitude in the absence of feedback is reduced by more than 50%, and shot to shot variation is reduced.

The resulting mode frequency spectra for no-feedback and feedback with 180° and 270° phasing are plotted in Figure 7.11. Comparing with Figure 7.5, one can see that the rotation frequency of the dominant mode has been reduced from -8 kHz to -6 kHz. With slowed down plasmas, no mode suppression effects were observed at any phase. Figure 7.11 illustrates how different feedback phases result in changing amplification factors and rotation frequencies.

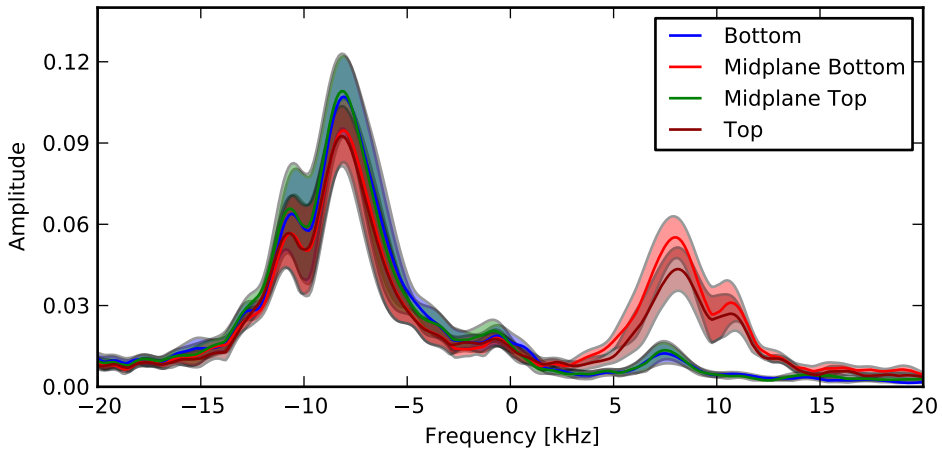

Figure 7.12: Frequency spectra of the $n = 1$ perturbations as measured the different sensor arrays without active feedback.

Strongest mode amplification by a factor of about 1.75 was now found at 270° phase. At this phase, mode rotation is also increased from -6 kHz to -6.8 kHz. If the phase is changed by 90° to 180°, amplification is reduced to 1.2, and rotation frequency decreases from -6 kHz to -5 kHz. Feedback gain in these experiments was 494. In experiments without bias probe, this gain resulted in strong excitations of slow -1.4 kHz modes (cf. Figure 7.8). With the bias probe present, no such excitation could be observed at all.

Frequency spectra of biased plasmas are also found to have more pronounced peaks that non-biased plasmas both with and without active feedback.

These results confirm that the effects of the bias probe on the plasma are very complex and not yet fully understood.

7.6 Positive/Negative Rotation Frequencies

One feature visible in all frequency spectra that has so far not been discussed is that peaks at negative frequencies are almost always accompanied by peaks of slightly reduced amplitude at the corresponding positive frequencies.

Based on the data shown so far, one could conclude that this is a real effect, and that the time evolution of the perturbation is more complex than simple rotation. However, there is evidence that peaks at positive frequency are actually measurements artifacts. Figure 7.12 shows the frequency spectra measured by the individual sensor arrays (without active feed-

back). Evidently, the peaks at positive frequencies exist only in the bottom-midplane and top arrays. If this were a real physical result, it would mean that the time evolution of the $n = 1$ perturbation alternates along the poloidal angle: at the bottom, the perturbation rotates in one direction, at the lower midplane, there is an additional perturbation of the same structure, but rotating in the opposite direction, at the upper midplane, there is pure rotation again, and at the top, there is again a superposition of two opposite rotations. Clearly, this is not a physically probable situation.

Instead, the positive frequency peaks are most likely caused by the sensor arrays having different sensitivities when measuring the sine and cosine components of the perturbation. Consider again the case of an idealized perturbation rotating at a frequency ω_0. The perturbed fields $B(\phi)$ at fixed poloidal angle θ_0 for such a perturbation are

$$B(\phi) = A\cos(n\phi + \omega_0 t + \delta) \tag{7.2}$$

which can be written as

$$B(\phi) = A\cos(n\phi)\cos(\omega_0 t + \delta) + A\sin(n\phi)\sin(\omega_0 t + \delta) \tag{7.3}$$

Multiplying both sides by either $\cos(n\phi$ or $\sin(n\phi)$ and integrating over ϕ this becomes

$$\int_0^{2\pi} \cos(n\phi)\, B(\phi)\, d\phi = A\cos(\omega_0 t + \delta)/\pi =: A_\mathrm{c}(t) \tag{7.4}$$

$$\int_0^{2\pi} \sin(n\phi)\, B(\phi)\, d\phi = A\sin(\omega_0 t + \delta)/\pi =: A_\mathrm{s}(t) \tag{7.5}$$

Ideally, the sensors measure exactly $B(\phi)$ at discrete positions. The integral then becomes a sum which can be carried out to calculate $A_\mathrm{c}(t)$ and $A_\mathrm{s}(t)$. The measured frequency spectrum then becomes

$$\mathscr{F}\left[A_\mathrm{c}(t) + i\,A_\mathrm{s}(t)\right] = A\,e^{i\delta}\delta(\omega - \omega_0) \tag{7.6}$$

as expected for the idealized case. However, if there is some misalignment or calibration problem, the measured values for A_c and A_s will differ from the true values $A\cos(\omega_0 t + \delta)$

and $A\sin(\omega_0 t+\delta)$ by some factors α and β. In this case, the fourier transform becomes

$$\begin{aligned}\mathscr{F}\left[A_\mathrm{c}(t)+iA_\mathrm{s}(t)\right] &= \mathscr{F}\left[\alpha A\cos(\omega_0 t+\delta)+i\beta A\sin(\omega_0 t+\delta)\right]\\ &= Ae^{i\delta}\frac{\alpha-\beta}{2}\delta(\omega-\omega_0)\\ &+ Ae^{i\delta}\frac{\alpha+\beta}{2}\delta(\omega+\omega_0)\end{aligned} \tag{7.7}$$

In other words, any measurement error that affects A_c and A_s in the same way will only affect the amplitude in the frequency plot. However, if the measurement errors in A_c and A_s differ (e.g., if there is $n=1$ error in the sensor alignment or gain), the frequency spectrum will show an additional peak at $-\omega_0$, which the amplitude proportional to the difference between α and β.

While not a physically existing effect, the positive frequency peaks are nevertheless seen by the control system and thus impair its frequency detection. It is therefore likely that control system performance can be further improved by more careful sensor calibration.

7.7 Comparison with Previous Results

The HBT-EP tokamak has recently undergone a major update that involved replacing both shell, control coils and magnetic sensors. While there is plenty of data available from prior feedback control experiments [38, 32, 34], none of it has therefore been obtained under the same conditions. Nevertheless, a comparison with previous results is worthwhile.

In its previous configuration [31], the HBT-EP shell consisted of toroidally alternating segments of aluminum and stainless steel. Steel segments were 0.2 cm thick with an L/R time of 300 µs. Aluminum segments were 1.4 cm thick and had a characteristic L/R time of 60 ms, but were retracted to minimize coupling to the plasma during feedback experiments. 40 control coils were mounted on the outer toroidal edges of the stainless steel shells in a 4x10 grid, and each toroidal control coil array was paired with a corresponding array of 5 poloidal field sensors located in between the control coils. The feedback algorithm then extracted the amplitude of the $n=1$ fourier components in each sensor array to generate a corresponding perturbation in the control coils. Control processing was implemented in 4 independently running FPGA modules.

In the first set of experiments performed by Klein et al. [38], the control algorithm applied

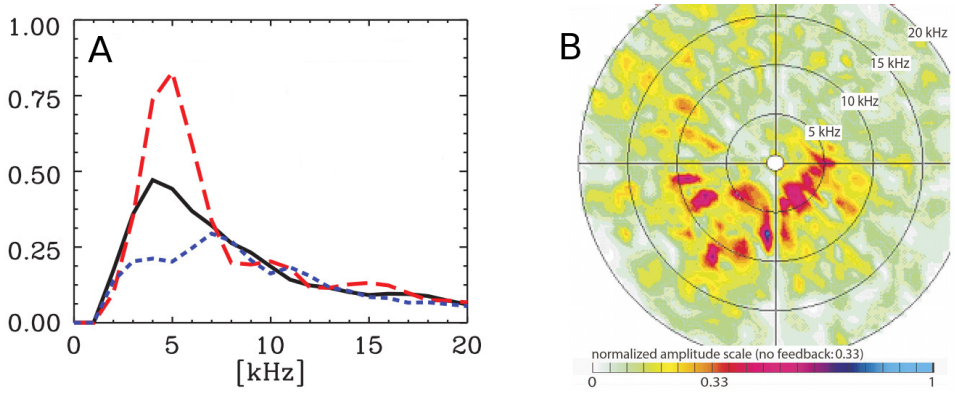

Figure 7.13: Results from prior feedback control experiments at HBT-EP. A: frequency spectrum of perturbations in poloidal field sensors with no feedback (black), suppressing (blue) and amplifying (red) feedback. Figure from Hanson et al. [33]. B: frequency spectrum of $m = 3$ rogowski signals with feedback at different phases (angular coordinate). Figure from Klein et al. [38].

a fixed gain and phase shift to the amplitudes measured by the sensors. Additional analog filters were used to reduce the effect of frequency dependent responses in analog system components. This algorithm ran with a sampling period and total latency of 10 µs, and a summary of the experimental results is shown in part B of Figure 7.13. Here the angular coordinate indicates the phase offset applied by the control system, and the contours reflect the frequency spectrum of poloidal field perturbations measured with an $m = 3$ rogowski coil. In plasmas without feedback, the spectrum peaks at 4 kHz with a normalized amplitude of 0.33. With phase offsets around 90°, this mode was effectively suppressed to the noise level.

The next iteration of feedback experiments performed by Hanson et al. [32, 33] used the same sensors, actuators and control hardware, but extended the algorithm with an internal system model implemented as a Kalman filter. This filter eliminated high frequency noise in the $n = 1$ perturbations and reduced the required control power without affecting control performance. Analog filters were also replaced by digital filters, which allowed to tune the control systems for different rotation frequencies. Part A of Figure 7.13 highlights the most significant results of this work. It shows the frequency spectrum of poloidal field fluctuations (in this case measured by the poloidal sensors that were also used for real-time control) with and without feedback. Depending on the feedback phase, the Kalman-filter-based control algorithm was able to amplify and suppress the amplitude at 4 kHz by up to 50% while keeping the overall spectrum mostly unchanged. The algorithm was run with a sampling period of 5 µs and a total latency of 10 µs. Measurements with artificially increased latency showed

a gradual decrease of suppression performance, with no visible suppression at latencies exceeding 30 µs [31].

The adaptive algorithm presented in this thesis was designed to further improve on the Kalman-based system. It addresses two main points:

1. Firstly, a significant challenge in previous experiments has been to correctly compensate for the frequency dependent phase and gain responses of sensors and actuators. Even careful measurement of the transfer functions and the use of an optimized filter would result in a phase variation of about 90° and a gain variation of 15% for frequencies between 1 kHz and 10 kHz. The algorithm used for the experiments in this thesis therefore takes a different approach: instead of filtering individual signals to flatten the time-response function, the signals are taken as-is, and knowledge of the rotation frequency is used to change the toroidal phase of the control output. In other words, the compensation is done in space rather than in time. As described in Section 6.4, this reduces phase and gain variations to the noise level.

2. Secondly, the Kalman-filter-based system had several free parameters that needed to be tuned for the equilibrium and perturbation that was to be controlled. The system model required specification of mode growth rate and rotation frequency, and the optimal Kalman gain depended on two 2x2 matrices of noise covariances. To achieve the results shown in Figure 7.13, the optimal parameters had to be identified by scanning the parameter space, a lengthy procedure that required large numbers of shots. The adaptive algorithm presented in this thesis improves upon this by determining equilibrium and perturbation dependent parameters at run-time.

When comparing the results obtained with the adaptive algorithm running on the GPU-based control system with those from the FPGA and Kalman-filter-based system, one finds that the GPU-based system is able to suppress mode activity with similar or improved effectiveness: the dominant peak in the mode spectrum was lowered by up to 50% in previous experiments and up to 60% in the experiments for this thesis. Numerical comparisons of the total mode amplitude (i.e., the area under the frequency curve) could not be performed due to the lack of more detailed data from previous experiments, but a visual comparison of Figures 7.8 and 7.13 suggests similar improvements. The improvements in suppression efficiency are achieved despite the perturbations rotating about twice as fast, and the system latency being 60% higher, but also with twice as many sensors as the previous system had

available. More importantly however, while previous results were obtained after a careful optimization of the model parameters, the results presented in this thesis required no parameter scans (the suppressing phase of -100° was determined entirely from induced currents prior to feedback system activation).

Chapter 8

Comparison with Simulations

This chapter compares numerical simulations of feedback controlled plasmas with the experimental data presented in Chapter 7. There are two well known models for the behavior of magnetically confined plasmas within the framework of MHD. In the Boozer model [9, 7, 5, 6], the plasma is assumed to be in ideal MHD equilibrium at all times and time-dependent behavior arises from changing boundary conditions. The Boozer model allows calculations in toroidal geometry, but is limited to ideal MHD (i.e., negligible plasma resistance and viscosity). The Fitzpatrick-Aydemir model [17, 16] is able to take into account more complex plasma parameters like viscosity, but uses a simpler set equations called *reduced MHD* and can therefore not capture toroidal effects and interactions between perturbations with different poloidal mode numbers. Neither model alone is found to be accurate for HBT-EP, so a combination of the models is used to simulate the effects of feedback control on HBT-EP plasmas. Relating the results to the experimental observations requires to compare exponential time constants with time averaged, fluctuating mode amplitudes, so only qualitative comparisons are possible. Within the subset of features that are observed experimentally and can be represented in the model, most of the observations agree with the predictions. Both suppression and frequency response of the dominant -8 kHz mode are predicted correctly. The excitation of a slower, co-rotating mode is correctly predicted, but its frequency response found to be shifted by 90° between experiments and simulations.

8.1 The Boozer Model

The Boozer model [9] is a general formulation for the evolution of ideal MHD plasmas. Its validity depends on the disparity between the timescale on which ideal MHD evolution happens, and the timescale on which plasma evolution can be effectively observed and

controlled: if an ideal MHD equilibrium is perturbed, the plasma will respond to this perturbation on an Alfvénic timescale and either settle into a new equilibrium, or loose confinement and disrupt. In magnetically confined plasmas the Alfvén timescale is in the order of a few microseconds, so that any plasma of interest must, on average, be in ideal MHD force balance. Non-Alfénic, slower plasma evolution can then be understood as a transition of the plasma through a sequence of ideal MHD plasma equilibria that is caused by non-ideal effects and/or changing boundary conditions. In case of the RWM, these changes are due to the eddy currents in the walls surrounding the plasma, and thus the timescale of RWM evolution is determined by the characteristic L/R time of the wall..

8.1.1 Plasma

In the Boozer model, the plasma is assumed to always satisfy the MHD equilibrium equation,

$$\vec{j} \times \vec{B} = \nabla p \tag{8.1}$$

Mathematically, this equation defines a time-independent, constant plasma state. Physically, however, no plasma completely obeys the equations of ideal MHD, and the boundary conditions (in form of externally produced fields) for the above equation may be varying in time. Therefore the quantities in the Equation 8.1 are considered time varying, but constrained in their values such that Equation 8.1 is fulfilled at any point in time. The resulting evolution of the plasma state can be split into a constant, known solution (the *unperturbed state*), and a time-varying, 3D perturbation.

The 3D perturbation will change the flux surfaces, currents and pressure profiles of the unperturbed state. However, if the change in pressure is due to movement of the flux surfaces only, and thus remains unchanged when expressed in magnetic coordinates, the solution to the linearized Equation 8.1 is completely determined by specifying the perturbing external field. By virtue of Laplace's equation, this field is in turn completely defined by its normal component on any "control surface" that separates the plasma from the external field sources. For axisymmetric unperturbed equilibria, codes like IPEC [50] and MARS-F [41] can be used to find this solution for arbitrary external field distributions.

By introducing a set of orthonormal surface functions $f_i(\theta,\phi)$, the normal component of

any field can be expressed as a vector $\vec{\Phi}$ such that

$$\vec{B}(\theta,\phi,r_{\text{surface}})\cdot\hat{n}=\sum_i \Phi_i\, f_i(\theta,\phi) \tag{8.2}$$

The expansion coefficients of the normal field due to the perturbed plasma current will be denoted $\vec{\Phi}_{\text{p}}$. The expansion coefficients of the normal field due to the perturbing external currents will be denoted $\vec{\Phi}_{\text{x}}$. We furthermore define the *surface plasma current* $\vec{I}_{\text{p}}$ to be the surface current that would give rise to the same fields $\vec{\Phi}_{\text{p}}$ as the actual volume currents in the plasma. The relation between $\vec{\Phi}_{\text{p}}$ and $\vec{I}_{\text{p}}$ is given by an inductance matrix $\mathbf{L}_{\text{p}}$:

$$\vec{\Phi}_{\text{p}}=\mathbf{L}_{\text{p}}.\vec{I}_{\text{p}} \tag{8.3}$$

$\mathbf{L}_{\text{p}}$ is a purely geometrical quantity and can be calculated from Maxwell's equations.

Since Laplace's equation implies that the effect of the plasma on any external structures is completely specified by the normal magnetic field due to the plasma on the control surface and, also, the effect of external currents on the plasma is completely specified by giving the normal field due to the external currents on the control surface, we define the plasma reluctance matrix $\mathscr{M}^{-1}$ as the mapping from any external field $\vec{\Phi}_{\text{x}}$ to the resulting perturbed currents $\vec{I}_{\text{p}}$:

$$\vec{I}_{\text{p}}=\mathscr{M}^{-1}.\vec{\Phi}_{\text{x}} \tag{8.4}$$

Treating the plasma as being in MHD equilibrium thus reduces the partial differential equation 8.1 to the system of linear, algebraic equations 8.4. This approximation is valid if the slow, sub-Alfvénic evolution of the plasma is dominated by changes in the boundary conditions of equation 8.1. This is the case if the plasma is MHD stable and the effects of external fields are dominating over internal dynamics not modeled by MHD, but also if the plasma is MHD unstable, but the growth rate of the instability limited by the resistance and proximity of a conducting wall. In the latter case, the evolution of the plasma is called a RWM and could be frozen at any time by surrounding the plasma with a perfect conductor, which is equivalent to prescribing the boundary conditions for Equation 8.1.

8.1.2 Wall and Control Coils

In order to obtain a time-dependent system equation, the plasma model (Equation 8.4), has to be extended with a model of the external conductors (the "wall") that are associated with

the changing boundary conditions.

Distributed currents in the wall can be represented as mesh currents $\vec{I}_w$ using a finite element model. Control coils are treated as additional discrete wall elements $\vec{I}_c$ with a potentially non-zero, externally supplied voltage $\vec{V}_c$. Control coils and wall elements together form the external currents $\vec{I}_x = (\vec{I}_w, \vec{I}_c)$ with driving voltages $\vec{V}_x = (0, \vec{V}_c)$. The generated external fields Φ_x on the control surface can be calculated from Maxwell's equations as

$$\vec{\Phi}_x = \mathbf{M}_{px}.\vec{I}_x \tag{8.5}$$

This is the defining equation for the mutual inductance matrix $\mathbf{M}_{px}$.

In the absence of a plasma, the external currents would evolve according to an ordinary circuit equation,

$$\mathbf{L}_x.\frac{d\vec{I}_x}{dt} + \mathbf{R}_x.\vec{I}_x = \vec{V}_x \tag{8.6}$$

Here $\mathbf{L}_x$ and $\mathbf{R}_x$ are the self inductance and resistance matrices of the external currents.

8.1.3 Complete System

From Equations 4.11 and 8.5, the plasma response is given by

$$\vec{I}_p = \mathscr{M}^{-1}.\mathbf{M}_{px}.\vec{I}_x \tag{8.7}$$

and an equation for the complete system is obtained by extending Equation 8.6 by the fluxes that arise from the plasma currents $\vec{I}_p$:

$$(\mathbf{L}_x + \mathbf{M}_{xp}.\mathscr{M}^{-1}.\mathbf{M}_{px}).\frac{d\vec{I}_x}{dt} + \mathbf{R}_x.\vec{I}_x = \vec{V}_x \tag{8.8}$$

Defining

$$\mathbf{A} = -\left(\mathbf{L}_x + \mathbf{M}_{xp}.\mathscr{M}^{-1}.\mathbf{M}_{px}\right)^{-1}.\mathbf{R}_x \qquad \text{and} \tag{8.9}$$

$$\mathbf{B} = -\mathbf{A}.\mathbf{R}_x^{-1} \tag{8.10}$$

Equation 8.8 is transformed into the standard form,

$$\frac{d\vec{I}_x}{dt} = \mathbf{A}.\vec{I}_x + \mathbf{B}.\vec{V}_x \tag{8.11}$$

The total field at the control surface, which determines the shape of the plasma, is

$$\begin{aligned}\vec{\Phi} &= \vec{\Phi}_x + \vec{\Phi}_p = (\mathbf{L}_p.\mathscr{M}^{-1} + \mathbf{1}).\mathbf{M}_{px}.\vec{I}_x \\ &=: \mathbf{P}.\mathbf{M}_{px}.\vec{I}_x\end{aligned} \tag{8.12}$$

P is called the permeability matrix [7]. To specify the complete state of the system, it is thus sufficient to give the wall currents $\vec{I}_x$.

8.1.4 Measurements

A magnetic field sensor measures the magnetic flux (or its time derivative) through the sensor area. Since in the Boozer model all the magnetic fields arise from known, discrete currents, measurements are simply linear combinations of the currents with the coefficients being determined by the positions of the sensors and currents. Mathematically, the fluxes $\vec{\Phi}_s$ through a set of sensors are given by

$$\vec{\Phi}_s = \mathbf{M}_{sp}.\vec{I}_p + \mathbf{M}_{sx}.\vec{I}_x \tag{8.13}$$

Here $\mathbf{M}_{sp}$ and $\mathbf{M}_{sx}$ are the mutual inductance matrices between the sensors and the plasma modes and external currents. In practice, sensors will of course also measure the static equilibrium fields. However, for simulation purposes this field is irrelevant and assumed to subtracted from the measurements.

Using Equation 8.7, the plasma state can be eliminated from Equation 8.13 so that

$$\begin{aligned}\vec{\Phi}_s &= \left(\mathbf{M}_{sp}.\mathscr{M}^{-1}.\mathbf{M}_{px} + \mathbf{M}_{sx}\right).\vec{I}_x \\ &=: \mathbf{C}.\vec{I}_x\end{aligned} \tag{8.14}$$

8.1.5 Validation of Model Assumptions: The Dissipative Shell

Equation 8.11 together with Equation 8.14 defines a complete dynamical model for the combined plasma wall system. A very important feature of this model is that the state of the system is completely defined by the state of the external currents, $\vec{I}_x$. This is because it is assumed that the plasma has no memory, and just instantaneously reacts to the fields produced by the external currents. The implication is that if the wall is moved away from the plasma towards infinity, the model will predict a completely quiescent plasma as there is no

interaction with the wall that could produce any perturbations.

Physically, this is of course not what happens. As the wall is being moved away, the evolution of an ideal plasma becomes Alfvénic. This evolution cannot be predicted by the model, because it was derived explicitly for time scales longer than the Alfvén time. For a non-ideal plasma, non-ideal effects like resistance and viscosity will start to increasingly dominate the plasma evolution. These effects prevent the evolution from reaching Alfénic speeds, but instead introduce a time dependence in the plasma response (Equation 8.4), thus also invalidating the predictions of the model.

Distinguishing between the case of a plasma that is being stabilized by external conductors, and a plasma that is unstable but appears stable because it does not couple sufficiently strong to the external conductors is thus obligatory for any simulations. The VALEN code [3] addresses this problem by pretending that the plasma is enclosed by an additional, conformal wall called the *dissipative shell*, which does not exist in the actual experiment. If the resistance of the dissipative shell is chosen suitably high, it will not affect any simulations for which the model assumptions are valid, since any currents induced in the shell will decay away very quickly and thus be negligible compared to the currents in the actual wall. However, if the model assumptions are not valid because of insufficient coupling between plasma and physical wall, there will be no currents in the physical wall. In this case the currents in the dissipative shell will determine the system evolution. The predicted system evolution will not be meaningful, because the dissipative shell does not physically exist. However, by virtue of the dissipative shell affecting the system evolution, it is then established that the model assumptions are not appropriate for the simulated situation and that a different model is needed.

Putting it differently, *if a simulation result is changed when adding (or removing) a hypothetical dissipative shell with high resistance, the model assumptions are invalid.* In this case, *neither the result with nor the result without dissipative shell is meaningful* [6].

It has been argued that the dissipative shell may be used as a blackbox model of an unknown dissipation mechanism inside the plasma by fitting the resistivity of the shell to observed growth rates and rotation frequencies. In this case the system model indeed appears to give reasonable dispersion curves. However, the resulting dynamics will still be arbitrary, as there is no prescription for choosing the mutual inductances between the hypothetical dissipative shell and the real wall, sensors and control coils.

8.2 The Fitzpatrick-Aydemir Model

A second model for the evolution of magnetic perturbations in a combined plasma-wall system is the Fitzpatrick-Aydemir model [17, 16]. In contrast to the Boozer model, the Fitzpatrick-Aydemir model is based on a simplification of the MHD equations that imposes assumptions on the ratio of minor to major radius and toroidal to poloidal field strength. The resulting set of equations is called *reduced MHD* [57, 39]. Reduced MHD eliminates many fast phenomena that happen in the direction parallel to the magnetic field lines, but preserves dynamics in the plane perpendicular to them. For this reason, the Fitzpatrick-Aydemir model can not capture toroidal effects and interactions between perturbations with different poloidal mode numbers. However, it is able to take into account more complex plasma parameters like resistance, inertia and viscosity.

The idea of the Fitzpatrick-Aydemir model is to partition the plasma into multiple layers, so that different aspects can be neglected in each layer. In the core, the plasma is treated as obeying ideal MHD. The core is surrounded by an inertial layer and a skin current layer, that are both assumed to be very thin. As the name indicates, the plasma in the inertial layer is assumed to have non-negligible inertia that damps the speed of any distortions produced by the core. The skin current layer captures the effects of viscous dissipation and resistivity and provides a sink for the energy of favorable distortions. The skin current layer is surrounded by a vacuum region, which is in turn enclosed by a resistive, continuous wall. With this representation, the following equations can be derived for the time evolution of the total poloidal flux at the plasma surface (Ψ_a), the total poloidal flux at the wall (Ψ_w), a flux due to active control coils behind the wall (Ψ_c) and the toroidal plasma rotation rate Ω:

$$\frac{d^2\Psi_a}{dt^2} + (\nu_* - 2i\Omega_\phi)\frac{d\Psi_a}{dt} + \left[(1-\kappa)(1-md) - \Omega_\phi^2 - i\nu_*\Omega_\phi\right]\Psi_a = \sqrt{1-(md)^2}\Psi_w \tag{8.15a}$$

$$S_*\frac{d\Psi_w}{dt} + (1+md)\Psi_w = \sqrt{1-(md)^2}\Psi_a + 2md\Psi_c \tag{8.15b}$$

$$\frac{d\Omega_\psi}{dt} + \nu_*(\Omega_\phi - \Omega_\phi^{(o)}) = -\nu_*\Omega_\phi|\Psi_a|^2 \tag{8.15c}$$

All variables in the above equations are normalized. Mauel [42] has rewritten this model in terms of a different set of parameters, and extended it to take into account control coils that affect not just the flux at the wall but also the flux at the plasma surface. This formulation

reads:

$$\frac{1}{\gamma_{\mathrm{MHD}}^2}\frac{d^2\Psi_\mathrm{a}}{dt^2} - \left(\alpha + 2i\frac{\Omega^2}{\gamma_{\mathrm{MHD}}^2}\right)\frac{1}{\Omega}\frac{d\Psi_\mathrm{a}}{dt} + \left(1 - s + i\alpha - \frac{\Omega^2}{\gamma_{\mathrm{MHD}}^2}\right)\Psi_\mathrm{a} = \frac{1}{\sqrt{c}}(\Psi_\mathrm{w} + c_\mathrm{f}\Psi_\mathrm{c}) \tag{8.16a}$$

$$\frac{d\Psi_\mathrm{w}}{dt} + \frac{\gamma_\mathrm{w}}{1-c}\left(\Psi_\mathrm{w} - \sqrt{c}\Psi_\mathrm{a}\right) = \gamma_\mathrm{w}\Psi_\mathrm{c}\left(1 - \frac{c_\mathrm{f}}{1-c}\right) \tag{8.16b}$$

The meaning of the parameters is as follows:

- Ω is the toroidal plasma rotation frequency
- c and c_f are parameters describing the coupling between the plasma and the wall, and the plasma and the control coils
- γ_{MHD} determines the time scale $\gamma_{\mathrm{MHD}}\sqrt{1-s}$ on which a mode would grow in the absence of dissipation.
- γ_w is the eddy current decay time of the wall
- s determines the stability of the plasma
- α determines the torque exerted on the plasma by the perturbation and can be calculated from a dissipation rate ν as

$$\alpha = \frac{-\nu\Omega}{\gamma_{\mathrm{MHD}}^2} \tag{8.17}$$

The crucial difference between Equations 8.12 and Equations 8.4 is that the former is written in terms of the distance between plasma and wall (encapsulated in the d parameter), while the latter is written in terms of the plasma stability s.

In either formulation, the magnetic field at the wall is given by

$$\vec{B}_\mathrm{w} = \nabla(\psi_\mathrm{a} + \psi_\mathrm{w}) \times \vec{z} \tag{8.18}$$

so that a poloidal field sensor at the wall measures a field of

$$B_\mathrm{w} = -\frac{\partial}{\partial r}\left(\psi_\mathrm{a} + \psi_\mathrm{w}\right) \tag{8.19}$$

Mauel has also expressed this in terms of the coupling coefficient c as:

$$B_\mathrm{w} = \frac{2\sqrt{c}}{1-c}\phi_\mathrm{a} - \frac{1+c}{1-c}\phi_\mathrm{w} \tag{8.20}$$

8.3 Parameter Matching

The advantage of the Boozer model is that, if it is applicable, it provides a description of the plasma wall system that takes into account the exact geometry of the wall, control coils and sensors and allows arbitrary deformations of the plasma surface, but the absence of dissipative effects makes the this model inapplicable to the HBT-EP tokamak: as reported in Chapter 6 and Chapter 7, magnetic perturbations in HBT-EP rotate with frequencies around 8 kHz. When solving Equation 8.11 without feedback ($V_x = 0$), none of the system eigenmodes comes close to such rotation frequencies. This is because the torque that can be exerted by a perturbation is limited by the plasma response (as determined by the eigenvalues of the permeability matrix **P**) to this perturbation. Any equilibrium that is sufficiently unstable for its permeability matrix to support the required amount of torque is also strongly affected by the addition of a dissipative shell, i.e. no longer in the validity region of the Boozer model.

The Fitzpatrick-Aydemir model is able to produce modes with the observed rotation frequencies. However, it is not expected to give accurate dynamic results either, because the segmented wall of HBT-EP is very different from the continuous, axisymmetric wall assumed by this model.

In order to obtain a more appropriate model of plasma dynamics of HBT-EP, the dissipation features of the Fitzpatrick-Aydemir model can be combined with the realistic representation of the HBT-EP wall provided by the Boozer formulation. This combined model has been used for dynamical simulations of HBT-EP plasmas in the past and was found to be reasonably accurate [43] and consistent with high dissipation caused by neoclassical flow damping [18]. The combined model will therefore be used for the simulations presented in this chapter, and an outline of its derivation is given below.

In order to obtain a more appropriate model of plasma dynamics of HBT-EP, the the dissipation features of the Fitzpatrick-Aydemir model can be extended with the with realistic MHD eigenvalue and plasma profiles. The method for this was proposed by Mauel, and first used in [Fitzpatrick and Bialek, 2006]. This study identified neoclassical flow damping to best represent the high-dissipation observed in HBT-EP. The single-helicty dynamics for the RWM and the viscously-damped kink-mode used previously to model mode dynamics in HBT-EP [Mauel, et al., 2004] is consistent with full eigenvalue analysis provided that the dissipation is sufficiently high. Because the single-helicity dynamical equations provide a reasonably accurate model, they will also be used for the simulations presented in this chapter. An outline of he procedure is given below.

For following derivations, it is assumed that the basis functions f_i (cf. Equation 8.2) are chosen to be complex (with the modulus indicating the toroidal orientation). The first step is reduce the Boozer model to a single plasma mode. This is done by diagonalizing the permeability matrix $\mathbf{P}$, and then projecting $\mathbf{P}$, $\mathbf{M}_{\mathrm{px}}$ and $\mathbf{M}_{\mathrm{xp}}$ onto the eigenvector with the largest eigenvalue. $\mathbf{P}$ then becomes a complex scalar. This results in a dynamic equation for a plasma that permits only a single rigid deformation, but still interacts with a finite element model of the wall.

In the second step, the wall currents are reduced to a single distribution that will give rise to the dominant system eigenmode. This is achieved by plugging the projected permeability matrix into Equation 8.12 and Equation 8.9 to get a reduced system evolution matrix $\mathbf{A}$. This matrix is again diagonalized, and Equation 8.8 projected onto the dominant eigenvector. This will also reduce $\mathbf{L}_{\mathrm{x}}$ and $\mathbf{R}_{\mathrm{x}}$ to scalars. The Boozer model has thus been reduced to a single rigid plasma perturbation interacting with a single wall current distribution, i.e. the assumption that the Fitzpatrick-Aydemir model is based on.

In the last step, the Fitzpatrick-Aydemir equations in the formulation by Mauel are simplified for the case of no dissipation. The reduced Fitzpatrick-Aydemir and Boozer models are still not expected to agree, but by considering limiting cases for the plasma-wall coupling and the plasma stability, the corresponding quantities in both models can be identified and one finds that [42, 9, 7]

$$c = \mathbf{L}_{\mathrm{p}}^{-1}.\mathbf{M}_{\mathrm{pw}}.\mathbf{L}_{\mathrm{w}}^{-1}.\mathbf{M}_{\mathrm{wp}} \tag{8.21}$$

$$c_{\mathrm{f}} = \mathbf{1} - \mathbf{L}_{\mathrm{w}}^{-1}.\mathbf{M}_{\mathrm{wc}}.\mathbf{M}_{\mathrm{pc}}^{-1}.\mathbf{M}_{\mathrm{pw}} \tag{8.22}$$

$$s + i\alpha = -\mathbf{P}^{-1} \tag{8.23}$$

$$\gamma_{\mathrm{w}} = \mathbf{L}_{\mathrm{w}}^{-1}.\mathbf{R}_{\mathrm{w}} \tag{8.24}$$

With these identifications, the parameters describing the geometry of the wall and the coupling of wall and control coils to the toroidal plasma can be transplanted into the Fitzpatrick-Aydemir model.

The resulting linear system obeys the following equation:

$$\frac{d}{dt}\begin{pmatrix}\psi_{\mathrm{a}}\\ \psi_{\mathrm{w}}\end{pmatrix} = \begin{pmatrix}\frac{1-s+i\alpha}{\alpha}\Omega & -\frac{\Omega}{\alpha\sqrt{c}}\\ \frac{\gamma_{\mathrm{w}}\sqrt{c}}{1-c} & -\frac{\gamma_{\mathrm{w}}}{1-c}\end{pmatrix}.\begin{pmatrix}\psi_{\mathrm{a}}\\ \psi_{\mathrm{w}}\end{pmatrix} + \begin{pmatrix}-\frac{c_{\mathrm{f}}\Omega}{\alpha\sqrt{c}}\\ \gamma_{\mathrm{w}}\frac{1-c_{\mathrm{f}}}{1-c}\end{pmatrix}.\psi_{\mathrm{c}} \tag{8.25}$$

8.4 Plasma Parameters

To determine the values of c, c_f and γ_w, a non-rotating, axisymmetric equilibrium with an edge q of 2.9 and normalized beta of 1.4 was calculated with plasma parameters typical for HBT-EP. The DCON code was used to calculate the corresponding plasma permeability matrix. The multi-mode VALEN code was used for computing mutual inductances, self inductances and resistances using the DCON provided plasma modes and a geometric model of the HBT-EP wall and control coils. The system was then reduced to single-mode as explained above. With the full set of large control coils, this resulted in the following values

$$c = 0.261 \tag{8.26}$$

$$c_f = 0.752 - 0.274i \tag{8.27}$$

$$\gamma_w = 3.42\ \mathrm{ms}^{-1} \tag{8.28}$$

The remaining parameters were chosen based on a combination of estimation from previous experiments and calculation of simplified cases. The mode rotation frequency Ω was set to -8 kHz based on the observed rotation frequencies in the absence of feedback. γ_{MHD} was chosen based on cylindrical calculations and measurements presented in Cates [12].

The plasma stability and torque parameters s and α can generally be obtained from the reduced permeability matrix $\mathbf{P}$. However, the DCON code that has been used to calculate $\mathbf{P}$ does not take into account toroidal plasma rotation, so that permeability matrix always gives $\alpha = 0$. For this reason, DCON computations have been used to determine the shape of the plasma mode (which is contained in the c and c_f parameters), but s and α have been chosen based on previous measurements [54, 55] which determined that $s \approx 1$ and $\nu/\gamma_{MHD} \approx 4.5$, so that $\alpha = 2.26$.

8.5 Numerical Method

For numerical calculations, the system model Equation 8.25 was extended by an ODE that relates the flux from the control coils ψ_c to the voltage output from the control system, V:

$$\frac{d\psi_c}{dt} = -\tau\psi_c + V \tag{8.29}$$

τ was chosen as 100 µs, the measured L/R time of the combined coil-amplifier system at 10 kHz. The detailed response of the coil-amplifier system that was measured in Chapter 6 is thus not included in the model. This is not expected to affect the accuracy of the simulations, because the linear system model can only produce solutions rotating at fixed frequencies. Not modeling the frequency dependence of the coil/amplifier response thus results at most in a constant gain or phase offset.

The state vector of the extended system model thus consists of 3 elements. The system was discretized with the correct sampling period of 4 µs and a zero-order hold approximation for the voltage input, i.e. the discrete system is exact if the voltage input updates instantaneously every 4 µs. The control output was related to the sensor measurements (Equation 8.19) with a feedback law of the form

$$V[n] = g e^{i\delta} \left(\frac{2\sqrt{c}}{1-c} \psi_{\mathrm{a}}[n-4] - \frac{1+c}{1-c} \psi_{\mathrm{w}}[n-4] \right) \tag{8.30}$$

i.e., the total poloidal field at the wall is multiplied by a gain g and rotated toroidally by a phase δ. The total processing latency of 16 µs was handled by the introduction of additional "memory" states, resulting in a total system dimension of 7.

Simulations were performed without artificial noise, so that the growth rates and rotation frequencies for different feedback phases δ and feedback gains g could be computed by eigenvalue analysis of the closed system.

8.6 Comparison Criteria

The system model that has been used for simulations is linear. Any solution is thus exponentially growing or decaying at a fixed rate, and rotating at a fixed frequency. The perturbations that have been observed experimentally are fluctuating in time, and rotate at varying frequencies. For this reason, the comparison between simulation results and experiments can be done only qualitatively. This means that comparisons will be made between the direction of relative changes in growth rate (for the simulation) and time averaged fluctuation amplitude (for experimental data), while the absolute values (including the sign) will be disregarded.

Also, since the theoretical model is limited to a single rigid deformation, any effects that are connected with the control system tracking multiple modes at different poloidal locations can not possibly be reproduced in simulations. With these restrictions, the following observations can potentially be compared to theoretical predictions:

- At -100° phase the feedback system reduces the mode amplitude and keeps the rotation frequency unchanged.
- At -10° phasing, mode amplitudes are reduced (but to a lesser extent than at -100°), and mode rotation frequency increases.
- At -190°/170° phasing, mode amplitudes are reduced (but to a lesser extent than at -100°), and mode rotation frequency decreases.
- At -280°/80° phase, neither mode amplitude nor mode rotation frequency are affected by the feedback system.
- If the feedback gain at -100° exceeds a specific threshold, mode suppression efficiency no longer increases, and instead a slower mode is excited.
- The frequency and amplitude of the slow mode increases from -10° to -280°/80°, but drops from 80°/-280° to -10°. (-0.7 kHz at -10°, -1.3 kHz at -100°, -3.6 kHz at -190°, and -6.1 kHz at -280°).

8.7 Simulation Results

Figure 8.1 shows the results of simulating feedback at different phases. Independent of the phase, simulations always result in one quickly rotating mode (around 8 kHz, plotted in green) and two slowly rotating modes (plotted blue and red). Dotted lines indicate the growth rates and rotation frequencies in the absence of feedback. Because of the scale, the change of frequency with phase is difficult to see in Figure 8.1. Figure 8.2 therefore shows the deviation of the frequency from its mean value. Figure 8.3 shows the growth rates for increasing feedback gain at a feedback phase of -100°.

From the figures, the following observations can be made and are mostly in agreement with the experimental observations listed above:

- At -100° phase, the fast mode is suppressed and its rotation frequency is unchanged. Two slow modes are being amplified without changes in rotation frequency.
- At -10° phase, the fast mode is suppressed (but to a lesser extent than at -100°) and its rotation frequency increased. Two slow modes are being amplified. The co-rotating mode is slowed down, and the counter-rotating mode is sped up.

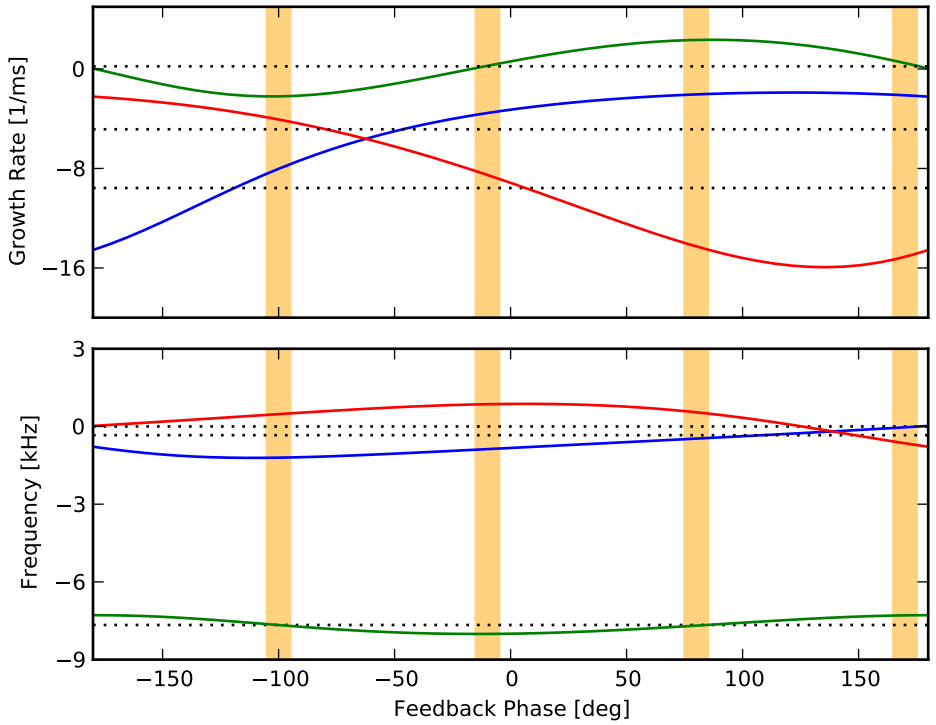

Figure 8.1: Growth rates and rotation frequencies for different feedback frequencies at 0.09 gain. Dotted lines indicate the growth rate and frequency in the absence of feedback. (Simulation results, shaded regions indicate the phases that have been tested experimentally).

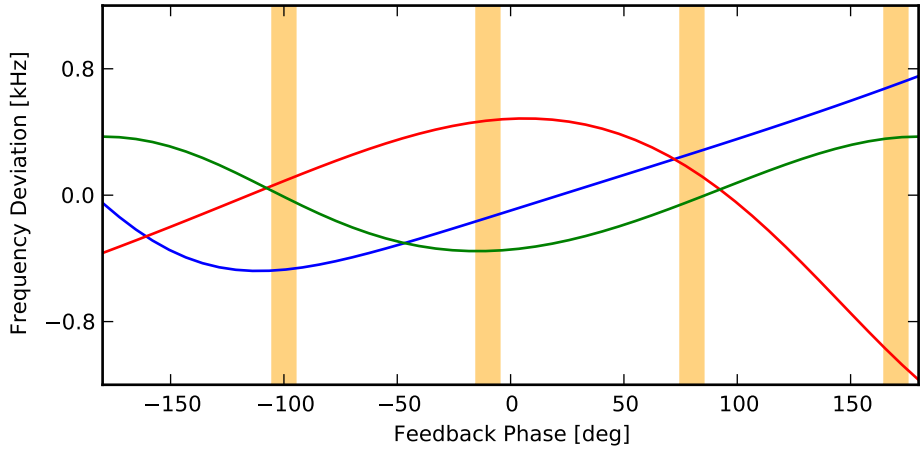

Figure 8.2: Relative changes in rotation frequency for different feedback phases (compared to the average over all feedback phases). (Simulation results, shaded regions indicate the phases that have been tested experimentally).

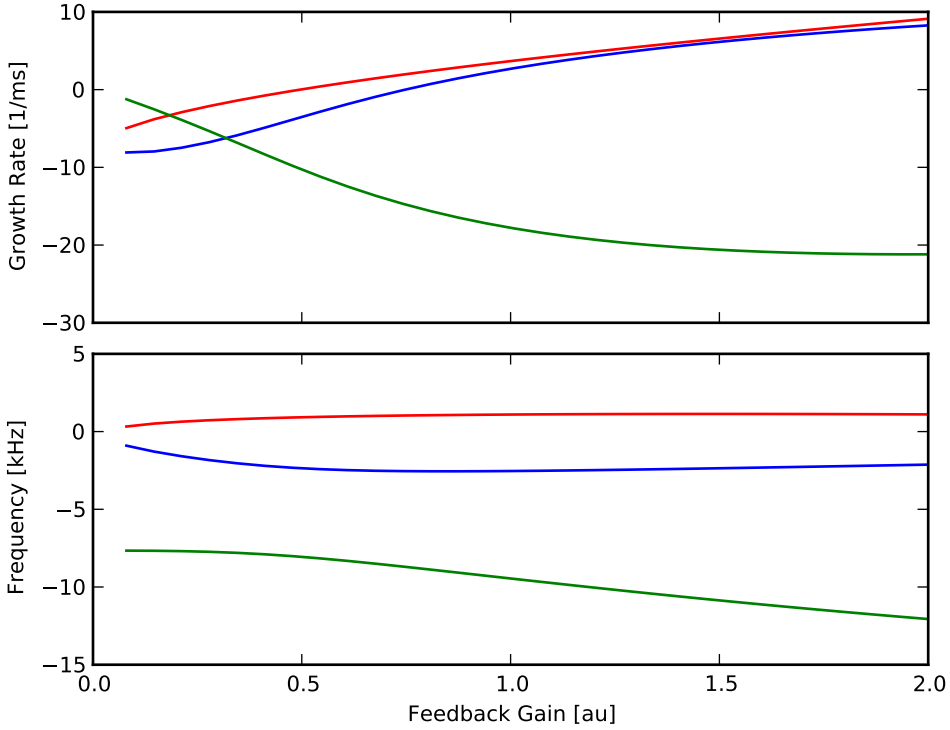

Figure 8.3: Growth rates and rotation frequencies with different feedback gains at -100° phase. (Simulation results).

- At -190°/170° phase, the fast mode is suppressed (but to a lesser extent than at -100°) and its rotation frequency decreased. The co-rotating slow mode is suppressed and sped up, and the counter-rotating slow mode is amplified and its frequency passes through zero.

- At -280°/80° phase, the fast mode is amplified and its rotation frequency unchanged. The co-rotating slow mode is amplified and slowed-down, and the counter-rotating slow mode is suppressed and slowed-down.

- With increasing feedback gain at -100° phase, the fast mode is increasingly suppressed and both slow modes become increasingly amplified. At a threshold gain of roughly 0.8, both slow modes become unstable (the co-rotating mode slightly earlier than the counter-rotating one) and the suppression of the fast mode levels out.

While the excitation of the slow counter rotating mode with increasing gain, has not been observed in experiments, this is most likely a result of the low frequency of this mode. Since the control system uses a high-pass filter to subtract equilibrium fields from sensor signals, it is likely that the counter-rotating mode is completely invisible to the control system and thus never excited (note that in the absence of feedback control, the plasma produces just two co-rotating modes, while the third mode is a decoupled L/R decay of the control coil currents without rotation).

There is, however, apparent disagreement between simulations and experiment in two aspects. While it is not surprising that the lack of any positive feedback in experiments at 80° is not predicted by the model, there is also disagreement in the frequency response of the slow mode. In experiments, the slow mode had a minimum frequency at -10° phase and a maximum at 170° phase. In simulations, the frequency of the co-rotating slow mode peaks at -100°, and has its minimum at 80°, i.e. the frequency response is shifted by 90° in phase compared to experiments. It is interesting to note that this means that the relative change in frequency of the co-rotating mode is still in agreement with the simulation at all phases except at -280°/80°. The reason for this disagreement is unclear, but could hopefully be elucidated by additional experiments that test the frequency response with a finer phase resolution.

With the exception of the phase shift in the frequency responses of the slow mode, the experimental observations are thus in qualitative agreement with the simulation results. In particular, the excitation of a co-rotating slow mode with increasing gain is predicted correctly,

and the effects of changing feedback phase on the rotation frequency of both the dominant mode are reproduced.

Chapter 9

Summary and Conclusions

9.1 Key Achievements

- A GPU-based control system with microsecond latencies and deterministic real-time performance has been designed. A patent application for this design has been filed.
- The design was implemented in a new plasma control system for the HBT-EP tokamak and shown to achieve latencies down to 8 µs and sampling periods down to 4 µs.
- A novel method for separating sensor signals into perturbed and equilibrium components that is not based on temporal smoothing was developed. The method was used as a benchmark for the implementation of a real-time separation algorithm and for all post-shot data analysis.
- An adaptive algorithm for the control of multiple rotating magnetic perturbations was developed.
- The algorithm was implemented and tested on HBT-EP's new control system. Performance immediately matched best results from fine-tuned previous systems without the need for any parameter scans. The amplitude of the dominant rotating perturbation was suppressed to 40% of the value without feedback.
- Numerical simulations of the feedback effects on the combined plasma-wall system were performed. Within the constraints imposed by the theoretical model, most results were found to be in agreement with experimental observations.

9.2 Summary of Results

In this thesis, we have presented a novel architecture for a plasma control system that employs a Graphics Processing Unit (GPU) for control computations. The two cornerstones of this architecture are the use of direct, peer-to-peer data transfers between AD/DA modules and GPU memory, and the full delegation of program control from CPU to GPU for the entire control algorithm. A patent application for this system has been filed by Columbia Technology Ventures.

The architecture was used to build a new control system for the HBT-EP tokamak. The new system uses a NVIDIA GeForce GPU and D-TACQ AD/DA modules providing a total of 96 input and 64 output channels with a resolution of 16-bit. In tests, sampling times down to 4 µs and latencies down to 8 µs were achieved. As expected, the GPU was shown to offer significantly more computing power and improved real-time performance than CPU-based solutions. HBT-EP's new control system is the first reported use of GPUs for data processing with microsecond latencies.

One of the main research areas of HBT-EP is the physics of rotating, 3D magnetic perturbations. Such perturbations have been studied and controlled at HBT-EP for several years, resulting in a continuous refinement of the used control systems and algorithms. The control system presented in this thesis continues this tradition. Limitations in the number of input/output channels have in the past necessitated the use of several independent control processors that each processed subsets of the available measurements to generate control signals for a subsets of the available control coils. The system described in this thesis removes this limitation and enables integration of all available feedback sensor measurements into one system state, which can then be used to generate control signals for all available control coils. Furthermore, the new GPU system allows a much faster and simpler development cycle for the control software than the previous FPGA-based system.

Complementing the new control system, a new control algorithm was developed and implemented as well. This algorithm improves upon previous work by using an adaptive system model that also supports the control of multiple perturbations at the same time. In the past, a scan of the model parameters was required to determine the values resulting in best control performance. With the new algorithm, the model parameters are deduced at run-time. This feature is made possible in large parts by the higher computing power offered by the GPU-based system. Other features of the new algorithm are the ability to compensate for arbitrary frequency dependent responses in analog control components and the playback

of arbitrary waveforms for open-loop experiments.

The combination of new control system and control algorithm was also tested experimentally. To allow comparison with earlier work, the algorithm was set up for the control of a helical $n = 1$ perturbation with unknown poloidal spectrum. It was found that in the time window where the plasma major radius is less than 92 cm, the system is able to suppress the time averaged amplitude of the dominant 8 kHz mode by up to 60%. This is comparable to the best results that have been achieved for 4 kHz modes in the past. Varying the feedback phase was shown to reduce suppression efficiency and slow down or speed up the mode rotation frequency. Suppression efficiency was found to increase with feedback gain up to a threshold value. Above this value, further increases in gain did not result in increased suppression efficiency but instead excited an additional 1.4 kHz mode. Mode amplification was only observed in the time window where the plasma major radius is above 92 cm. Here amplification by a factor of up to two was demonstrated.

Experimental observations were also compared with theoretical models. Initial plans to do simulations using the multi-mode Boozer model had to be changed, as it was found that the lack of dissipation dynamics precluded it from being able to model the high frequency perturbations that are typically observed in HBT-EP plasmas. Instead, the computed couplings between plasma, walls and control coils were used to introduce some effects of the actual experimental geometry into the formally cylindrical Fitzpatrick-Aydemir model. This required to reduce the representable perturbations to a single rigid perturbation of the plasma surface, and the possible current distributions in the wall to a single pattern with a fixed resistance and inductance. Within these restrictions, qualitative agreement was found between experimental observations and simulation results. Quantitative comparisons were hampered by the necessity to compare exponential growth rates (that are predicted by theory) to time averaged fluctuation amplitudes (that are observed in experiments).

9.3 Implications

Most of the work described in this thesis has been targeted at designing a new generation of control systems for magnetically confined plasmas. However, the presented control system architecture is general enough to be used for any sort of control problem that requires low latency and high computing power. The control system that has been implemented for HBT-EP could be taken as-is, connected to a different set of sensors and actuators and be

used for controlling an entirely different process.

This thesis has also demonstrated that reasonably complex algorithms can be run exclusively on GPU cores which dramatically improves real-time performance. While much work in the field of scientific computing is being done on making the use of a GPU fully transparent (e.g. by having the compiler automatically distribute programs between CPU and GPU), we have shown that there are also significant advantages in the opposite approach, where code is written exclusively for execution on the GPU. However, there are still wide ranges of functionality that, even though implemented in the GPU, can only be accessed from a thread running on the CPU. These restrictions on GPU code impose a limit on the complexity of algorithms that can run completely on the GPU and will hopefully continue to be relaxed.

The importance of feedback control for achieving high performance fusion plasmas is well established, and control of single resistive wall modes (RWM) has been shown to allow access to higher performance regimes (above the *no-wall* limit) than possible without feedback [36, 27, 48]. Access to even higher regimes will require control of multiple modes [43, 36]. Additionally, potentially beneficial effects of stable 3D perturbations have been reported [10, 15]. The implications of this thesis for the control of high performance fusion plasmas are therefore twofold.

Firstly, it has been shown that control of multiple perturbations, using even large numbers of sensors and control coils is possible, and can be performed with microsecond latencies and sampling intervals. This opens up the possibility of increasing plasma performance from the no-wall limit all the way to the ideal wall limit (at which growth rates become Alfvénic) using feedback control.

Secondly, being able to process large numbers of measurements and control outputs in a single control processor can be used to implement more sophisticated control algorithms, and to extend feedback control from suppressing specific 3D perturbations to actively controlling the 3D shape of the plasma. In particular, the implementation of optimal control algorithms with large numbers of states is expected to significantly improve feedback performance [37], and the ability to control the 3D shape of the plasma may improve confinement and transport [51]. While not described in this thesis, a second algorithm for use with HBT-EP's new control system has been developed as well. This algorithm works with a much larger system model that is based on the multi-mode Boozer model and thus allows representation of the full 3D structure of the plasma. Using modern control theory, the control matrices are derived from performance indices and are not limited to driving the plasma towards axisymmetry, but

allow arbitrary, 3D reference states. Problems with the applicability of the Boozer model to HBT-EP prevented experimental testing of this algorithm. However, this problem is not likely to occur in larger experiments with more sophisticated control of the axisymmetric MHD equilibrium. For such experiments, GPU-based control systems like the one presented in this thesis should thus be able to not only suppress instabilities, but also allow keeping the plasma in specific 3D equilibrium states, which may improve plasma confinement, transport and stability.

9.4 Directions for Further Research

This thesis has presented the design and implementation of a novel, GPU-based control system. The system was also tested experimentally with an adaptive control algorithm that reproduced the best feedback results from past experiments without any parameter optimization other than a rough phase and gain scan. Qualitative agreement between experimental results and a theoretical model was found as well. Nevertheless, the work performed for this thesis should be understood as merely establishing the foundation for future experimental research on advanced feedback control. It is expected that the new system will be flexible and powerful enough to support experiments over multiple iterations of control algorithms and sensor-actuator combinations.

The results that have been obtained so far already suggest several topics for future investigation:

- The effects of feedback have so far only been investigated in the frequency domain of the $n = 1$ mode that has been controlled. It would be worthwhile to also investigate the effect that feedback has on the overall shape of the plasma. For individual shots, the spatial structure is best determined by biorthogonal decomposition (as explained in Levesque [40]). However, to allow a systematic analysis and isolate the effects due to feedback, some way needs to be found to average these results over multiple shots.

- Experiments for this thesis have been performed with the minimum possible sampling period to make the control system output as smooth as possible. However, this has been done by sacrificing control system latency. An important question is if feedback results would improve or worsen if the control system were set up to minimize latency at the expense of sampling period.

- The only parameters that were varied for the feedback experiments presented in this thesis were the base feedback gain and phase. The control algorithm, however, has several more parameters that can be adjusted and whose effects on feedback performance should be investigated. Most notably, the derived rotation frequencies have so far only been used to compensate for digital latency and analog response, but not for the elimination of sensor noise.

- Furthermore, the performed experiments considered only four $n = 1$ perturbations. This was done because prior experiments established that control of this perturbation is possible, and it allowed comparison of the feedback efficiency. With the functionality of the system established, future experiments should investigate the possibility of controlling different mode combinations. In particular, trying to track the modes that are determined by biorthogonal decomposition would be worthwhile.

- HBT-EP has been designed with different control coil sets of varying plasma coverage. So far, only the largest set of control coils has been used. The effects of control coil coverage on feedback efficiency are especially important because the amount of space that is available for control coils in reactor-scale experiments will be very limited. Testing feedback efficiency with different sets of control coils is therefore an important task that HBT-EP experiments are especially suited for.

- The adaptive algorithm that has been presented in this thesis has been designed empirically based on the performance and limitations of previous feedback experiments and the observed dynamics in uncontrolled plasmas. However, there is also extensive literature on the theory of adaptive control. In particular, Sun et al. [58] have proposed an adaptive control algorithm for resistive wall modes that is based on the extended least squares system identification method. Implementing such a more advanced algorithm and testing its performance experimentally would give important information about the advantages that can be gained from further refinement of feedback algorithms.

- Finally, there is a stark contrast between the fluctuating amplitudes that are observed in experiments and the exponential growth or decay that is predicted by the theory used in numerical simulations. This was found to limit comparisons to qualitative behavior. However, with the number of comparable criteria being relatively low, many apparent agreements could actually be due to chance with reasonable probability. More confidence could thus be placed in results that can be compared not just qualitatively

but also quantitatively. Developing a theoretical model that includes some mechanism for mode amplitudes to saturate would thus significantly improve the usefulness of any feedback simulations.

Appendices

Appendix A.

Control System User's Guide

This chapter contains instructions for operation of the GPU-based control system installed for the HBT-EP tokamak. It does not discuss any details that are not required for day-to-day operation, and is meant both as a reference for users and as a further illustration of the capabilities and usability of the system.

A.1 Hardware

The host system is located on the *caliban* computer in the south rack. This system runs the same Ubuntu Linux installation as the rest of the HBT-EP computers. In particular, it automatically synchronizes to the reference installation on spitzer on every boot, so any changes performed locally do not persist over reboots.

Caliban is not directly integrated into the internal HBT-EP network. It is connected to the Columbia network with a separate fiber-optical converter. Connection to the internal network is established with a VPN. The VPN software is *tinc*.

The GPU is installed in the computer case. Due to space limitations, it is water cooled. This means that the case may not be used in any orientation, as in some orientations air bubbles may prevent cooling water from reaching the processor core.

The D-TACQ digitizer (ACQ196) and analog output boards (AO32CPCI) are housed in a separate CPCI chassis that is also installed in the south rack. The CPCI chassis provides both power and a CPCI connection between the boards, and also contains a CPCI to PCI bus extender. However, *the CPCI chassis is only used as a power supply* and neither the internal CPCI bus nor the bus extender are used. Instead, the D-TACQ boards communicate over rear transition modules. The used rear transition modules are also manufactured by D-TACQ and called *RTM-T*. They provide a separate PCI-Express cable connection for each CPCI board.

Caliban houses corresponding PCI-E cable adapters, and each RTM-T connects to its own PCI-E adapter.

A.2 GPU Drivers

The system uses the official NVIDIA developer GPU drivers. In order to support automatic recompilation for newer kernels and allow switching between different drivers, the drivers are installed as a Debian package and use the Debian Kernel Module System (DKMS). The package and driver source can be found in */usr/local/src/debian-packages/nvidia-graphics-driver-dev-** on spitzer. Upgrade and package creation is done according to the standard Debian packaging procedures, described e.g. in *www.debian.org/doc/manuals/maint-guide/*, so the driver package can be created with *debuild* and installed with *dpkg -i*. Compilation for the installed kernels happens automatically at installation. A manual recompilation can be enforced with *dkms build -m nvidia-current -v <module-version> -k <kernel-version>* followed by *dkms install -m nvidia-current -v <module-version> -k <kernel-version>*.

A.3 A-D/D-A Drivers

Since both ACQ196 and AO32CPCI modules use RTM-Ts to connect to the host computer, only one driver is needed to communicate with either device. The source for this RTM-T driver is located in */usr/local/src/RTM-T* on spitzer. The code is managed in a Mercurial repository to allow merging of local customisations with newer driver versions from D-TACQ. The unmodified D-TACQ source is tracked in the *upstream* branch. Local changes are mostly for compatibility with more recent Linux kernels and better integration with the Debian system setup.

The RTM-T kernel module is automatically compiled and installed for every new kernel by a script in */etc/kernel-pkg/postinst.d/*. It should be noted that the D-TACQ boards take some time to boot up. If the RTM-T driver is loaded by the host computer too early, this results in a confusing error message.

When the RTM-T driver has been loaded correctly, a new directory */dev/rtm-t.** will be created for every connected RTM-T.

A.4 Compiler and Libraries

The GPU kernel code can be compiled with either NVIDIA's nvcc or PathScale's ENZO. The former is installed in */opt/cuda*, the later in */opt/enzo*.

If nvcc is used for compilation, there are two libraries for communication with the NVIDIA driver: the standard CUDA library, or gdev. The gdev source code is located in */usr/local/src/gdev*, and the compiled library and header files are installed into */opt/gdev*.

The control system code resides in */opt/hbt/control* and can be compiled with *make -f <Makefile>*. Different make files are available to compile with different compilers and runtime libraries: *Makefile.cuda* for nvcc, *Makefile.gdev* for nvcc with gdev and *Makefile.enzo* for ENZO.

A.5 A-D/D-A Initialization

Prior to control system operation, the D-TACQ digitizers and analog output modules need to be switched to real-time operation. This is done with the script */opt/hbt/control/init_rtm-t.sh*. In addition to preparing real-time operation, this script also sets up signal routing for the ACQ196: master clock and trigger signals are read from the LEMO connector on the ACQ196. Divided clock and trigger is distributed to the AO32CPCIs via the internal PXI bus, and the divided clock furthermore forwarded to the rear I/O connector on the RTM-T. The signal routing of the AO32CPCIs is currently hardcoded in the RTM-T driver[1].

A.6 Preprogrammed Operation

Preprogrammed operation is implemented in the *do_awg* program. When started, it reads the clock divisor from the *CLOCK_DIV* environment variable and configures the ACQ196 accordingly. It then opens the file *awgdata.dat* and waits for the trigger signal to arrive. Once the trigger has arrived, on every (divided) clock tick a new vector of output samples is read from the data file and the analog output updated accordingly.

The recommended way to create *awgdata.dat* is to use the *prep_awg.py* Python script. It contains helper routines that know the mapping from output channels to control coils and can be used for a high-level description of the required wave form. For example, a waveform

[1] Yes, this is terrible and should be fixed as quickly as possible.

with *SAMPLES* data points that produces a toroidally rotating field with (m,n) helicity can be generated by the following code:

```
1     ccoils = [ 'FB%02d_C%d' % (i,j)
2                for j in np.arange(4)+1
3                for i in np.arange(10)+1 ]
4
5     cc_indices = np.empty(len(ccoils), dtype=np.int16)
6     for (i, name) in enumerate(ccoils):
7         cc_indices[i] = cc_map[name] - 1
8
9     phi = np.asarray([ cc_info['%s_%s' % (x, coil_size)].phi
10                       for x in ccoils ])
11    theta = np.asarray([ cc_info['%s_%s' % (x, coil_size)].theta
12                         for x in ccoils ])
13    times = np.arange(SAMPLES) * CYCLE_TIME
14    waveform = np.cos(m * theta + n * (phi + FREQUENCY * times))
```

A.7 Feedback Operation

Feedback control is implemented in the *do_fb* program and configured in the *fbsettings.py* file. Changes to *fbsettings.py* require a recompilation of *do_fb*, as some settings are embedded into the GPU instruction stream.

fbsettings.py may be a full-fledged Python program. However, its only purpose is to initialize several variables that define the behavior of the control system. The different variables are explained below.

AI_DIVISOR

This variable specifies the factor by which the external reference clock should be divided to obtain the sample clock for the control system.

CYCLE_TIME

This variable specifies the sampling period of the control system.

FB_LATENCY

This variable defines the total processing latency of the control system.

OFFSET_SAMPLES

When the control system is triggered, it assumes that the first *OFFSET_SAMPLES* samples are zero signals on all input channel. For every channel, the mean of these samples is subtracted from all future samples to compensate for constant digitizing offsets.

WAIT1_SAMPLES

After *OFFSET_SAMPLES* samples have been processed, the control system remains inactive for *WAIT1_SAMPLES* samples. Since the control system generally needs to be triggered before plasma breakdown (to gather the digitizing offsets), this setting allows deferring activation of the feedback system until the desired point in time.

WAIT2_SAMPLES

The control system begins tracking perturbations after *OFFSET_SAMPLES* + *WAIT1_SAMPLES* samples. However, the control output is not enabled for another *WAIT2_SAMPLES*. Delaying the output is sensible, because it will take the control system a while to lock on to the tracked perturbations.

FB_SAMPLES

This variable specifies the number of samples for which the feedback system will be fully active, i.e. track state and generate control output. After a total of *OFFSET_SAMPLES* + *WAIT1_SAMPLES* + *WAIT2_SAMPLES* + *FB_SAMPLES*, *do_awg* will terminate.

FIT_SAMPLES

This variable specifies the number of most recent samples that are used when fitting the sensor signals to a linear equilibrium signal.

FREQ_SAMPLES

This variable specifies the number of most recent samples that are used when fitting the phase evolution to a rotation frequency. For obvious reasons, *FB_SAMPLES* must be smaller than *WAIT2_SAMPLES*.

RC_TIME

This variable specifies the RC time to use when integrating raw sensor signals. The formula used by the control system is

$$B[n] = V[n] \cdot \text{RC\_TIME} + \sum_{i=0}^{n} V[i] \cdot \text{CYCLE\_TIME} \tag{A.1}$$

SENSORS

This variable should refer to a list of sensor names, in the order in which the *IN_MODE_MATRIX* (see below) expects signals to be arranged. The control system uses this variable to properly route control inputs, and to set up the proper calibration factors to account for different sensor gains and polarities.

IN_MODE_MATRIX

This variable should contain a real matrix. The matrix defines the mapping from integrated sensor signals to perturbation amplitudes. The number of input signals (i.e., number of columns) must equal the number of sensors defined in *SENSORS*. An arbitrary number of perturbations may be specified. Each perturbation is defined by two successive rows that project the sensor signals onto the cosine and sine component of the perturbation respectively.

CCOILS

This variable should refer to a list of control coil names, in the order in which the *OUT_MODE_MATRIX* (see below) expects output signals to be routed.

OUT_MODE_MATRIX

This variable should contain a real matrix. The matrix defines the mapping from perturbation amplitudes to control outputs. The number of perturbation amplitudes (i.e., number of columns) must be equal to the number of rows of *IN_MODE_MATRIX*. The number of output signals (i.e., number of rows) must be equal to the number of control coils defined in *CCOILS*.

COUPLED_FREQUENCIES

This boolean variable may be true or false. When true, all tracked perturbations are fitted to the same rotation frequency. When false, the rotation frequency will be tracked and fitted separately for every perturbation.

BASE_GAIN, GAIN_COEFF_x

These variables define the feedback gain. The effective control gain g is calculated as

$$g = \text{BASE\_GAIN} \cdot \sum_{i=x}^{2} \omega^x \, \text{GAIN\_COEFF\_}x \tag{A.2}$$

where ω is the rotation frequency. Therefore, *BASE_GAIN* times *GAIN_COEFF_o* is the basis gain, while the remaining variables specify the frequency dependent response of the analog actuators and ensure a flat frequency response of the entire system.

BASE_PHASE, PHASE_COEFF_x

These variables define the feedback phase. The effective control phase δ is calculated as

$$\delta = \text{BASE\_PHASE} + \sum_{i=x}^{3} \omega^x \, \text{GAIN\_COEFF\_}x \tag{A.3}$$

where ω is the rotation frequency. Therefore, *BASE_PHASE* is the basis phase, while the remaining variables specify the frequency dependent response of the analog actuators and ensure a flat frequency response of the entire system. For historical reasons, *BASE_PHASE* has to be specified in degrees, while *PHASE_COEFF_x* are specified in radians.

FB_FREQ_DIVISOR

This variable specifies how often the effective phase is updated. Since the rotation frequency is assumed to change slowly, it is often not required to update the effective feedback phase and gain on every sample. In this case, *FB_FREQ_DIVISOR* may be set to a positive integer that indicates how many samples may be processed with the same phase and gain. Setting it to a value other than 1 may allow running with lower latency and/or lower sampling period.

MIN_MODE_AMP

If the amplitude of a perturbation is very low, the calculated toroidal phase may be meaningless and cause large errors when least squares fitting to determine the rotation frequency. For this reason, measurements with a total amplitude below *MIN_MODE_AMP* Tesla will not be included in the least squares fit.

A.7.1 Example configuration

The following is the feedback configuration that has been used for most of the experiments described in Chapter 7. For a feedback phase or feedback gain scan, only the *BASE_PHASE*/*BASE_GAIN* values are changed between shots.

```
1  import numpy as np
2  from scipy.linalg import block_diag
3  import math
4  from math import radians
5
6  AI_DIVISOR=4
7  CYCLE_TIME=1e-6 * AI_DIVISOR
8  FB_LATENCY=16e-6
9  MIN_MODE_AMP=0
10 FB_FREQ_DIVISOR=2
11 RC_TIME=0.0002
12
13 OFFSET_SAMPLES=int(1e-3 // CYCLE_TIME)
14 WAIT1_SAMPLES=int(1.8e-3 // CYCLE_TIME)
15 WAIT2_SAMPLES=int(1.2e-3 // CYCLE_TIME)
16 FB_SAMPLES=int(4.0e-3 // CYCLE_TIME)
```

```
17 FIT_SAMPLES=int(1.24e-3 // CYCLE_TIME)
18 FREQ_SAMPLES=int(0.4e-3 // CYCLE_TIME)
19 TOTAL_SAMPLES = OFFSET_SAMPLES + WAIT1_SAMPLES + WAIT2_SAMPLES + FB_SAMPLES
20
21 BASE_GAIN = 494
22 GAIN_COEFF_0 = 7.07222968e-01
23 GAIN_COEFF_1 = -2.82440155e-04
24 GAIN_COEFF_2 = 5.43023193e-08
25
26 BASE_PHASE = 100 # In degrees here, radians below!
27 PHASE_COEFF_0 = 0.68644227349326226
28 PHASE_COEFF_1 = 4.77028412e-04
29 PHASE_COEFF_2 = 3.96448411e-08
30 PHASE_COEFF_3 = 1.29360740e-12
31
32 SENSORS = []
33 CCOILS = []
34 IN_MODE_MATRIX = []
35 OUT_MODE_MATRIX = []
36 for i in xrange(4): # For each toroidal array
37     sensors = [ 'FB%02d_S%dP' % (j,i+1) for j in np.arange(10)+1 ]
38     ccoils = [ 'FB%02d_C%d' % (j,i+1) for j in np.arange(10)+1 ]
39
40     out_mtx = np.zeros((10,2), dtype=np.float)
41     out_mtx[:,0] = np.cos(2*math.pi*np.arange(10) / 9)
42     out_mtx[:,1] = np.sin(2*math.pi*np.arange(10) / 9)
43
44     in_mtx_i = out_mtx.copy()
45     # Remove broken sensors
46     for name in ('FB06_S2P', 'FB08_S4P', 'FB10_S2P', 'FB10_S3P', 'FB10_S4P'):
47         if name not in sensors:
48             continue
49         idx = sensors.index(name)
50         del sensors[idx]
51         in_mtx_i = np.delete(in_mtx_i, idx, 0)
52     in_mtx = np.linalg.pinv(in_mtx_i)
```

```
53
54      SENSORS += sensors
55      CCOILS += ccoils
56      IN_MODE_MATRIX.append(in_mtx)
57      OUT_MODE_MATRIX.append(out_mtx)
58
59  IN_MODE_MATRIX = block_diag(*IN_MODE_MATRIX)
60  OUT_MODE_MATRIX = block_diag(*OUT_MODE_MATRIX)
61
62  COUPLED_FREQUENCIES=True
63  COIL_SIZE='big'
```

Most of the settings should be straightforward to understand. However, the calculation of the input and output matrices may warrant some additional explanation.

The above configuration makes use of the fact that the configuration file may be a full program on its own and generates the input/output matrices programmatically. The *for* loop iterates over the four toroidal arrays. For each toroidal array, the sensors and control coils forming the array are appended to the *SENSORS* and *CCOILS* variables. The rows of the output mode matrix for each toroidal array are then defined as the cosine and sine functions. To generate the input matrix, columns corresponding to sensors that are known to be faulty are removed, and the remaining matrix is pseudo-inverted. At the end, the input/output matrices for the individual arrays are assembled as diagonal blocks of the full input/output matrices.

A.7.2 Running the System

Once the feedback system has been suitable configured, a feedback controlled shot can be taken by running the *make fb_shot* command in the */opt/hbt/control* directory. This command should be started when the toroidal bank begins to charge. It will read the current shot number from the tree, recompile *do_fb* if required, and then wait for 90 seconds. After 90 seconds, the *do_fb* program is executed and will wait for a trigger signal. Shortly afterwards, the toroidal bank will be fully charged and the system be triggered. After the feedback system has terminated, the used feedback settings will be copied into the file */opt/hbt/data/control/fbsettings_xxx.py*, where *xxx* is the current shot number. This allows looking up the exact feedback configuration of any past shot. In addition to that, the system also stores several data files in the same directory. These files contain the control

input, control output, and internal control system state and can be used for debugging or performance optimization.

Appendix B.

Analysis Scripts

This chapter explains usage of the different analysis scripts that have been developed for this thesis.

All scripts have been written in Python and can be called directly from the command line. The behavior of the scripts can be customized using command line parameters, and every script accepts a *–help* parameter that causes it to print out usage instructions. With the exception of *find_similar_shots.py*, all programs are located in the */opt/hbt/control* directory.

The following scripts are available:

find_similar_shots.py

This script implements the algorithm for the selection of shots with similar plasma evolution. It reads the full corpus of shots from standard input (separated by white space) and prints out the reduced set of similar shots. Plots of the full and final set can be generated with the *–plot* option.

group_shots.py

This script groups a list of shots into groups with identical feedback settings. Feedback settings for each shot are read from the files in */opt/hbt/data/control*. The list of shots is read from standard input and should be separated by white spaces. The script prints out the different shot groups in form of a Python dictionary that can directly as input for the different analysis scripts.

find_filter_settings.py

This script determines the optimal window length and polynomial degree for the real-time equilibrium subtraction algorithm.

compare_diagnostics.py

This script compares the mode amplitude and phase as measured by different diagnostics. The following diagnostics may be specified

SENSORS Uses the feedback sensor data from the MDSplus tree.

AI_STORE Uses the feedback sensor data as received and stored by the control system.

AO_STORE Uses the control system output as stored by the control system.

ACQ12 Attempts to read the control system output from the CPCI_12 digitizer in the MDSplus tree. This requires that the analog output has been wired for this in the analyzed shot.

CC Reads control coil currents from the MDSplus tree.

ens_mode_spectra.py

This script produces ensemble averaged frequency spectra. It needs to be called with two arguments. The first argument must be a Python file that defines the different shots that are to be used (such a file can be generated using *group_shots.py*). The second argument should be a feedback configuration file (like *fbsettings.py*) that defines the structure of the modes that are to be analyzed.

ens_mpair_spectra.py

This script takes the same arguments as *ens_mode_spectra.py*. It also generates ensemble averaged shots of frequency spectra, however, this script calculates the spectrum of the real *quadrature amplitude*, rather than the complex combination of the sine and cosine amplitudes.

ens_mpair_ampl.py

This script takes the same arguments as *ens_mode_spectra.py* and plots ensemble averages of the mode amplitude over time.

ens_mpair_mean_ampl.py

This script takes the same arguments as *ens_mode_spectra.py* and plots ensemble averages of the mean mode amplitude, i.e. amplitudes are averaged over shots and over time.

ens_mpair_phasediff.py

This script takes the same arguments as *ens_mode_spectra.py* and plots ensemble averages of the phase difference between the perturbations measured by the sensors and generated by the control coil currents.

ens_mpair_freq.py

This script takes the same arguments as *ens_mode_spectra.py* and plots ensemble averages of the rotation frequency against time. Frequencies are determined by the real-time algorithm, i.e. by polynomial fitting of the toroidal phase to a linear polynomial.

check_gpu_calcs.py

This script contains a second, independent implementation of the adaptive control algorithm in Python and is used to verify that the two implementations are in agreement. It expects a shot number as a parameter and reads its control input from this shot. The calculation results are then compared with the results of the real-time algorithm (which are stored in */opt/hbt/data/control*).

Appendix C.

Annotated GPU Source Code

This chapter contains the source code of the GPU kernel implementing the adaptive feedback control algorithm. The code has been annotated to highlight and describe important steps and optimization techniques where appropriate.

C.1 gpu_common.cu

This file defines common procedures that are required for any feedback algorithm.

```
1  #ifndef GPU_COMMON_H_
2  #define GPU_COMMON_H_
3
4  #include <stdint.h>
5
6  #ifdef USE_ENZO
7  #include <driver_types.h>
8  #include <cuda_runtime.h>
9  #include <cuda.h>
10 #endif
11
12 // From rtm-t-llcontrol.cpp
13 #define LLCV2_STATUS_TLATCH 6     /** TLATCH {11:0} */
14 #define TLATCH_OFFSET ((3 * 16) + LLCV2_STATUS_TLATCH)  /* 32 bit words */
15
16 // Special latch value for timeout
17 #define TLATCH_TIMEOUT 0xFFFFFFFF
```

The first two constants define the offset in the input buffer at which the current *latch time* can be found. The latch time is the number of clock ticks received so far, and expected to increase by the clock divider with every sample. The second value is a dummy latch time that indicates that the maximum waiting time was exceeded when waiting for a new input vector.

```
18 __device__ uint32_t wait_for_sample(volatile int16_t* ai,
19                                       uint32_t tlatch_old) {
20     uint32_t pollcount = 0;
21     uint32_t new12, old12;
22     const uint16_t mask = 0xfff;
23     volatile uint32_t* tlatch_ptr = ((volatile uint32_t*) ai)
24                                       + TLATCH_OFFSET;
25
26     old12 = tlatch_old & mask;
27     while ((new12 = *tlatch_ptr & mask) == old12)
28         if (pollcount++ - sinf(pollcount) > 1<<28)
29             return TLATCH_TIMEOUT;
30
31     if (old12 > new12)
32         return ((tlatch_old & ~mask) | new12) + (mask + 1);
33     else
34         return (tlatch_old & ~mask) | new12;
35 }
```

This method implements a busy loop that waits for a new input vector to arrive. It returns the current latch time. The code may appear complicated because of the need to handle wrap-around of the latch time (since only the 12 least significant bits of the time are send by the digitizer). In order to slow down the memory polling rate, the sine of the current polling attempt is computed once in every iteration.

```
36 #ifdef USE_ENZO
37 #include "enzo_util.h"
38 void start_kernel(int threads, CUdeviceptr ai, CUdeviceptr ao,
39                   CUdeviceptr mailbox, CUdeviceptr ai_store,
40                   CUdeviceptr mamp_store,
41                   CUdeviceptr mphase_store, CUdeviceptr mfreq_store,
42                   CUdeviceptr ao_store) {
```

```
43      kernel<<<1, threads>>>((volatile int16_t*) ai,
44                             (volatile int16_t*) ao,
45                             (uint32_t*) mailbox,
46                             (float*) ai_store,
47                             (float*) mamp_store,
48                             (float*) mphase_store,
49                             (float*) mfreq_store,
50                             (int16_t*) ao_store);
51  }
52
53  #endif /* USE_ENZO */
54  #endif
```

This method is only compiled and used when using the ENZO compiler. Since ENZO does not support loading the GPU kernel from a separate file, the kernel is started using the inline CUDA syntax.

C.2 gpu_fb.cu

```
1   #define __STDC_LIMIT_MACROS
2   #include <stdint.h>
3
4   #include "gpu_api.h"
5   #include "fbsettings.h"
6   #include "gpu_common.cu"
7
8   #define PI 3.1415926535897931
9   #define MAX(a,b) ((a) > (b) ? (a) : (b))
10  #define MIN(a,b) ((a) > (b) ? (b) : (a))
```

These are convenience macros. Defining *__STDC_LIMIT_MACROS* ensures that the *INT16_MAX* macro will be defined by *stdint.h*.

```
11  // Offsets of sensors in 96 element input vector
12  __device__ int ai_map[] = SENSOR_INDICES;
13
```

```
14 // Offsets of control coils in output vector
15 __device__ int ao_map[] = CCOIL_INDICES;
16
17 /* Offsets at which to start reading matrices
18  * (used to optimize away leading zeros)
19  */
20 __device__ int in_mtx_x0[] = IN_MTX_X0;
21 __device__ int out_mtx_x0[] = OUT_MTX_X0;
22
23 // Feedback matrices -- will be cached in shared memory
24 __device__ float IN_MTX[][SENSOR_COUNT] = IN_MODE_MATRIX;
25 __device__ float OUT_MTX[][MODE_COUNT] = OUT_MODE_MATRIX;
```

The macros on the right hand side are defined in *fbsettings.h* and loaded into arrays stored in GPU memory.

```
26 /* Matrix multiplication based feedback
27  *
28  * Needs to be called with max(SENSOR_COUNT, MODE_COUNT, CCOIL_COUNT)
29  * threads.
30  */
31 extern "C"
32 __global__ void kernel(volatile int16_t* __restrict__ ai,
33                        volatile int16_t* __restrict__ ao,
34                        uint32_t* __restrict__ mailbox,
35                        float* __restrict__ ai_store,
36                        float* __restrict__ mamp_store,
37                        float* __restrict__ mphase_store,
38                        float* __restrict__ mfreq_store,
39                        int16_t* __restrict__ ao_store) {
```

The different _store variables are allocated in GPU memory by the loader process and used to store the time evolution of analog input, mode amplitudes, frequencies and phases, and analog output. The *ai* and *ao* variables are pointers to the input and output buffers written to and read from by the digitizers and analog output modules (thus the *volatile* declaration). The *mailbox* array is used to communicate the GPU state to the loader after the kernel has terminated. The *__restrict__* modifier declares that none of the parameters are overlapping

and allows the compiler to make some optimizations based on that.

```
40      int tid = threadIdx.x;
41      int i, i_max;
42      __shared__ int32_t missed_samples;
43      __shared__ float ai_c[SENSOR_COUNT];
44      __shared__ float mode_amps[MODE_COUNT];
45      __shared__ uint32_t tlatch, tlatch_old;
46
47      volatile int16_t *my_ai;
48      if(tid < SENSOR_COUNT)
49          my_ai = ai + ai_map[tid];
50
51      volatile int16_t *my_ao;
52      if(tid < CCOIL_COUNT)
53          my_ao = ao + ao_map[tid];
54      *my_ao = 0; /* Reset output */
55
56  #ifndef GPU_NOSTORE
57      int16_t *my_ao_store;
58      if(tid < CCOIL_COUNT)
59          my_ao_store = ao_store + ao_map[tid];
60  #endif
61
62      int in_x0;
63      if(tid < MODE_COUNT)
64          in_x0 = in_mtx_x0[tid];
65
66      int out_x0;
67      if(tid < CCOIL_COUNT)
68          out_x0 = out_mtx_x0[tid];
```

In a GPU kernel, all threads execute the same program code. In order for different threads to work on different data, the predefined *tid* variable contains a sequential number that is different for every thread. This variable is here used to define pointers to the input signal, output signal and matrix rows that the different threads should process.

The *in_xo* and *out_xo* variables contain the index of the first non-zero column in the input and output matrices. This allows threads to skip the zero multiplications which results in significant speedups for block diagonal matrices.

```
69      /* Transpose and store in shared memory for faster access */
70      __shared__ float in_mtx[SENSOR_COUNT][MODE_COUNT];
71      __shared__ float out_mtx[MODE_COUNT][CCOIL_COUNT];
72      if(tid < MODE_COUNT)
73          for(i=0; i < SENSOR_COUNT; i++)
74              in_mtx[i][tid] = IN_MTX[tid][i];
75
76      if(tid < CCOIL_COUNT)
77          for(i=0; i < MODE_COUNT; i++)
78              out_mtx[i][tid] = OUT_MTX[tid][i];
```

As indicated in the comment, the input and output mode matrices are transposed and stored in shared memory for faster access.

```
79      /* These constants make sure that the mode phase is calculated
80       * correctly, no matter if the current thread calculates a
81       * sine or cosine amplitude */
82      float phase_off;
83      int sign;
84      if(tid < MODE_COUNT && tid % 2 == 0) {
85          sign = 1;
86          phase_off = 0;
87      }
88      else if(tid < MODE_COUNT && tid % 2 == 1) {
89          sign = -1;
90          phase_off = PI/2;
91      }
```

Since there is a separate thread for the computation of every mode amplitude, there are two threads working on every mode *pair*. Thus, when calculating the toroidal phase, the threads need to ensure that they use the same conventions for which mode is considered the sine, and which the cosine component. For efficiency reasons, the code is written such that every thread assumes that it is the cosine component, but the resulting phase for the sine modes is

flipped and shifted by pi/2 to achieve consistency.

```
92      if(tid == 0) {
93          missed_samples = -1; // First sample will always appear
94                               // missing since we don't know device's
95                               // latch time
96          tlatch = 0xdeadbeef;
97          ((volatile uint32_t*) ai)[TLATCH_OFFSET] = tlatch;
98          mailbox[MB_GPU_STATUS] = MB_GPU_RUNNING;
99      }
100
101     __syncthreads();
```

This code fragment initializes the latch timer to ensure that the first sample can be detected.

```
102     int sample_no;
103     int16_t itmp;
104     float sig_offset = 0;
105     float acc=0, tmp, tmp2;
106     float K=0, eq=0;
107     float Kw=0, w=0; /* w: average phase jump per sample */
108     float fb_phase, fb_gain, cos_v, sin_v;
109     float last_phase=0, phase_acc=0, phase_diff;
110     for (sample_no = 0; sample_no < TOTAL_SAMPLES; sample_no++ ){
111
112         // Wait for next sample
113         if(tid == 0) {
114             tlatch_old = tlatch;
115             tlatch = wait_for_sample(ai, tlatch_old);
116
117             if (tlatch != tlatch_old + AI_DIVISOR)
118                 missed_samples++;
119         }
120         __syncthreads();
121         if(tlatch == TLATCH_TIMEOUT) {
122             *my_ao = 0; /* Reset output */
123             mailbox[MB_GPU_STATUS] = MB_GPU_TIMEOUT;
124             mailbox[MB_GPU_LAST_SAMPLE] = sample_no;
```

```
125             mailbox[MB_GPU_MISSED_SAMPLES] = missed_samples;
126             return;
127         }
```

This code forms the first part of the main loop. The first thread (with thread id zero) polls the input buffer for new data. The tlatch time is compared with the expected value to determine missed input vectors. In case of a timeout, the GPU kernel exits. The *__syncthreads* call ensures that execution only continues when all threads have reached this synchronization point.

```
128         /* Determine offset */
129         if(sample_no < OFFSET_SAMPLES) {
130             if(tid < SENSOR_COUNT)
131                 sig_offset += ((float) *my_ai) * 10 / INT16_MAX;
132             if(sample_no == OFFSET_SAMPLES-1)
133                 sig_offset /= OFFSET_SAMPLES;
134             continue;
135         }
```

For the first *OFFSET_SAMPLES*, the control system accumulates the sensor signals to determine any digitalization offsets. Sensor signals arrive as signed 16 bit integers with the maximum corresponding to 10 V and are converted to floating point voltages for further calculations. Note that *sig_offset* is a thread local variable, so every thread holds the offset for its own associated input signal.

```
136         /* Integrate signals */
137         if(tid < SENSOR_COUNT) {
138             tmp = ((float) *my_ai) * 10 / INT16_MAX - sig_offset;
139             acc += tmp * CYCLE_TIME;
140             tmp = acc + tmp * RC_TIME;
141 #ifndef GPU_NOSTORE
142             ai_store[sample_no * SENSOR_COUNT + tid] = tmp;
143 #endif
144             ai_c[tid] = tmp;
145         }
```

After *OFFSET_SAMPLES*, the control system subtracts the previously determined offset and integrates the sensor signals. The integrated signals are stored in *ai_store* and cached in

shared memory (*ai_c*).

```
146         /* Determine mode amplitudes and apply equilibrium filter */
147         __syncthreads();
148         if(tid < MODE_COUNT) {
149             // Preload old samples.
150             asm("prefetch.global.L1 [%0];" ::
151                 'l'(mamp_store + (sample_no - FIT_SAMPLES)*MODE_COUNT));
152             i_max = MIN(in_x0+IN_MTX_XLEN,SENSOR_COUNT);
153             for(i=in_x0, tmp=0; i < i_max; i++)
154                 tmp += ai_c[i] * in_mtx[i][tid];
155 
156             mamp_store[sample_no * MODE_COUNT + tid] = tmp;
157             tmp2 = mamp_store[(sample_no - FIT_SAMPLES)*MODE_COUNT + tid];
158             K += tmp - tmp2;
159             eq += EQ_FIT_3 * K + EQ_FIT_1 * tmp - EQ_FIT_2 * tmp2;
160             mode_amps[tid] = tmp - eq;
161         }
162 
163         if(sample_no < OFFSET_SAMPLES+WAIT1_SAMPLES)
164             continue;
```

This code fragment implements converts from sensor signals to mode amplitudes and subtracts equilibrium components. The signals that are about to leave the window over which equilibrium components are fitted are prefetched, and the mode amplitudes are calculated. Equilibrium components are then updated using the auxiliary K quantity as described in Section 5.3.

```
165         /* Determine mode phases and estimate frequency */
166         __syncthreads();
167         if(tid < MODE_COUNT) {
168 #if MIN_MODE_AMP_NONZERO
169             // Preload old samples.
170             asm("prefetch.global.L1 [%0];" ::
171                 'l'(mphase_store + (sample_no - FREQ_SAMPLES)*MODE_COUNT));
172 
173             // Only consider phase if amplitude is reasonable
```

```
174             tmp2 = mode_amps[tid + sign * MPAIR_COUNT];
175             if(sqrtf(tmp*tmp + tmp2*tmp2) > MIN_MODE_AMP) {
176 #endif
177                 tmp = atan2f(sign * mode_amps[tid + sign],
178                              mode_amps[tid]) + phase_off;
179                 phase_diff = tmp - last_phase;
180                 last_phase = tmp;
181                 if(phase_diff > PI)
182                     phase_diff -= 2*PI;
183                 else if(phase_diff < -PI)
184                     phase_diff += 2*PI;
185
186                 // Skip over jumps in frequency estimation
187                 if(phase_diff > PI/2 || phase_diff < -PI/2)
188                     phase_diff = w;
189
190                 phase_acc += phase_diff;
191 #if MIN_MODE_AMP_NONZERO
192             }
193             /* Mode amplitude too small, estimate phase measurement
194              * based on current frequency estimate */
195             else {
196                 last_phase += w;
197                 if(last_phase > 2*PI)
198                     last_phase -= 2*PI;
199                 else if(last_phase < -2*PI)
200                     last_phase += 2*PI;
201                 phase_acc += w;
202             }
203 #endif
```

The above code calculates the toroidal phase and quadrature amplitude of every mode. If the mode amplitude is too small, the phase is instead estimated from the frequency and the phase of the previous sample. If the phase is found to have jumped by more than pi, an offset of 2pi is added based on the assumption that the mode must have completed a toroidal rotation.

```
204             mphase_store[sample_no * MODE_COUNT + tid] = phase_acc;
205
206             tmp2 = mphase_store[(sample_no - FREQ_SAMPLES)*MODE_COUNT + tid];
207             Kw += phase_acc - tmp2;
208             w += FREQ_FIT_3 * Kw + FREQ_FIT_1 * phase_acc - FREQ_FIT_2 * tmp2;
209
210 #if !defined(GPU_NOSTORE) || defined(COUPLED_FREQUENCIES)
211             mfreq_store[sample_no * MODE_COUNT + tid] = w;
212 #endif
213         }
214
215
216         if(sample_no < OFFSET_SAMPLES+WAIT1_SAMPLES+WAIT2_SAMPLES)
217             continue;
```

This code computes the continuous least squares fit of the phase history (*mphase_store*) to a constant rotation frequency using the auxiliary *Kw* variable.

```
218         /*
219          * Apply feedback phase
220          * */
221         __syncthreads();
222         if(tid < MODE_COUNT) {
223             if(sample_no % FB_FREQ_DIVISOR == 0) {
224 #if COUPLED_FREQUENCIES
225                 /* No need to average over both pair members, they
226                  * calculate phases and frequencies from the same
227                  * data */
228                 for(i=0, tmp=0; i < MODE_COUNT; i++)
229                     tmp += mfreq_store[sample_no * MODE_COUNT + i];
230                 tmp /= MODE_COUNT;
231 #else
232                 tmp = w;
233 #endif
234                 tmp /= (2*PI*CYCLE_TIME);
235                 tmp2 = tmp;
236                 fb_phase = (BASE_PHASE*PI/180
```

```
237                             + PHASE_COEFF_0 + tmp * FB_LATENCY * 2 * PI
238                             + PHASE_COEFF_1 * tmp);
239                 fb_gain = (BASE_GAIN*GAIN_COEFF_0 + BASE_GAIN*GAIN_COEFF_1 * tmp);
240                 tmp *= tmp2;
241                 fb_phase += PHASE_COEFF_2 * tmp;
242                 fb_gain += BASE_GAIN * GAIN_COEFF_2 * tmp;
243                 tmp *= tmp2;
244                 fb_phase += PHASE_COEFF_3 * tmp;
245
246                 cos_v = __cosf(fb_phase) * fb_gain;
247                 sin_v = __sinf(fb_phase) * fb_gain;
248             }
249
250             mode_amps[tid]
251                 = cos_v * mode_amps[tid]
252                 - sign * sin_v * mode_amps[tid + sign];
253         }
```

Here the phase and gain of the control output is calculated from base feedback phase, digital latency and analog response. The mode amplitudes are then rotated and scaled accordingly.

```
254         /* Calculate control coil output */
255         __syncthreads();
256         if(tid < CCOIL_COUNT) {
257             i_max = MIN(out_x0+OUT_MTX_XLEN, MODE_COUNT);
258             for(i=out_x0, tmp=0; i < i_max; i++)
259                 tmp += mode_amps[i] * out_mtx[i][tid];
260
261             /* Safeguard against extreme currents */
262             if(tmp > 5)
263                 tmp = 5;
264             else if (tmp < -5)
265                 tmp = -5;
266
267             itmp = (int16_t) (tmp * INT16_MAX / 10);
268             *my_ao = itmp;
269 #ifndef GPU_NOSTORE
```

```
270             *(my_ao_store + sample_no * AO_CHANNELS) = itmp;
271 #endif
```

This code updates the output buffer and stores the desired output for archival.

```
272         }
273     }
274
275     *my_ao = 0; /* Reset output */
276
277     if(tid == 0) {
278         mailbox[MB_GPU_MISSED_SAMPLES] = missed_samples;
279         mailbox[MB_GPU_LAST_SAMPLE] = sample_no;
280         mailbox[MB_GPU_STATUS] = MB_GPU_DONE;
281     }
282 }
```

Appendix D.

Shot Numbers

The following HBT-EP shot numbers have been used for the results presented in this thesis:

Latency Measurements

RAM Passthrough: 76515, 76516, 76517, 76518, 76519, 76520, 76521, 76522, 76523, 76524, 76525, 76526, 76527, 76528, 76529, 76530, 76531, 76532, 76533

GPU RAM Passthrough: 76534, 76535, 76536, 76537, 76538, 76539, 76540, 76541, 76542, 76543, 76544, 76545, 76546, 76547, 76548, 76549, 76550, 76551, 76552

GPU RAM Copy: 76553, 76554, 76555, 76556, 76557, 76558, 76559, 76560, 76561, 76562, 76563, 76564, 76565, 76566, 76567, 76568, 76569, 76570, 76571, 76573, 76575

Host RAM Copy: 76576, 76577, 76578, 76579, 76580, 76581, 76582, 76583, 76584, 76585, 76586, 76587, 76588, 76589, 76590, 76591, 76592, 76593, 76594, 76595, 76596, 76597, 76598

CPU Benchmark: 76638, 76639, 76640, 76641, 76642, 76643, 76644, 76645, 76646, 76647, 76648, 76649, 76650, 76651, 76652, 76653, 76654, 76655, 76656, 76657, 76658, 76659, 76660, 76661, 76662, 76663, 76664, 76665, 76666, 76667, 76668, 76669, 76670

GPU Benchmark: 76616, 76617, 76618, 76619, 76620, 76621, 76622, 76623, 76624, 76625, 76626, 76627, 76628, 76629, 76630, 76631, 76632, 76633, 76634, 76635, 76636, 76637

Amplifier Response Measurements

75814, 75815, 75816, 75817, 75818, 75819, 75820, 75821, 75822, 75823, 75824, 75825, 75826, 75827

Feedback Experiments

Campaign 1: 74775 74761 74762 74763 74764 74767 74768 74769 74770 74774 74776 74777 74780 74782 75225 75168 75171 75172 75173 75175 75176 75177 75178 75179 75187 75188 75189 75212 75230 75216 75220 75228 75229 75234 75240 75244 75248 75205 75213 75217 75221 75231 75235 75241 75245 75249 75206 75214 75218 75222 75223 75232 75237 75242 75246 75207 75215 75219 75224 75226 75238 75243 75247

Campaign 2: 75311 75317 75323 75324 75325 75328 75329 75330 75331 75332 75334 75335 75336 75337 75338 75339 75341 75342 75344 75346 75347 75348 75349 75350 75351 75352 75353 75355 75356 75357 75358 75359 75362 75363 75364 75365 75366 75367 75370 75371 75372 75373 75374 75377 75378 75379 75381 75382 75384 75385 75386 75390 75391 75392 75393 75394 75395 75397 75398 75399 75400 75401 75402 75403 75404 75405 75407

Campaign 3: 75436 75438 75439 75441 75442 75444 75450 75454 75455 75456 75457 75458 75460 75462 75463 75464 75465 75467 75468 75469 75470 75472 75501 75502 75505 75506 75507 75508 75509 75511 75513 75514 75517 75518 75519 75520 75523 75524 75525 75529 75530 75532 75333 75534 75535 75536 75537 75539 75540 75541 75542 75543 75544 75545 75546 75447 75552 75554 75559 75562 75563 75565

Campaign 4: 75869 75868 75867 75866 75863 75865 75907 75908 75909 75910 75911 75912 75913 75914 75915 75917 75918 75919 75920 75921 75922 75923 75924 75925 75926 75927 75928 75929 75930 75931 75932 75933 75934 75935 75936 75937 75938 75939 75940

Campaign 5: 75987 75989 75990 75991 75992 75993 75994 75995 75996 75997 75998 75999 76000 76001 76002 76003 76004 76008 76009 76010 76011 76012 76013 76014 76015 76017 76018 76019 76020 75863 75865 75866 75867 75868 75869 75907 75908 75909 75911 75914 75918 75920 75923 75927 75929 75936

Appendix E.

Bibliography

[1] Nadine Aubry, Regis Guyonnet, and Ricardo Lima. Spatiotemporal analysis of complex signals: Theory and applications. *Journal of Statistical Physics*, 64(3-4):683–739, August 1991. ISSN 0022-4715. doi: 10.1007/BF01048312. URL *http://www.springerlink.com/index/10.1007/BF01048312*.

[2] Paul. M. Bellan. *Fundamentals of Plasma Physics*. Cambridge University Press, 2006. ISBN 978-0521821162.

[3] J.M. Bialek, Allen H. Boozer, Michael E. Mauel, and Gerald A. Navratil. Modeling of active control of external magnetohydrodynamic instabilities. *Physics of Plasmas*, 8(5):2170, 2001. ISSN 1070664X. doi: 10.1063/1.1362532. URL *http://link.aip.org/link/PHPAEN/v8/i5/p2170/s1&Agg=doi*.

[4] Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schröder. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Transactions on ...*, pages 917–924, 2003. URL *http://dl.acm.org/citation.cfm?id=882364*.

[5] Allen H. Boozer. Equations for studies of feedback stabilization. *Physics of Plasmas*, 5(9): 3350, 1998. ISSN 1070664X. doi: 10.1063/1.873048. URL *http://link.aip.org/link/PHPAEN/v5/i9/p3350/s1&Agg=doi*.

[6] Allen H. Boozer. Feedback equations for the wall modes of a rotating plasma. *Physics of Plasmas*, 6(8):3180, 1999. ISSN 1070664X. doi: 10.1063/1.873558. URL *http://link.aip.org/link/PHPAEN/v6/i8/p3180/s1&Agg=doi*.

[7] Allen H. Boozer. Resistive wall modes and error field amplification. *Physics of Plasmas*, 10(5):1458, 2003. ISSN 1070664X. doi: 10.1063/1.1568751. URL *http://link.aip.org/link/PHPAEN/v10/i5/p1458/s1&Agg=doi*.

[8] Allen H. Boozer. Physics of magnetically confined plasmas. *Reviews of Modern Physics*, 76(4):1071–1141, January 2005. ISSN 0034-6861. doi: 10.1103/RevModPhys.76.1071. URL *http://link.aps.org/doi/10.1103/RevModPhys.76.1071*.

[9] Allen H. Boozer. Simplified multimode calculations of resistive wall modes. *Physics of Plasmas*, 17(7):072503, 2010. ISSN 1070664X. doi: 10.1063/1.3453706. URL *http://link.aip.org/link/PHPAEN/v17/i7/p072503/s1&Agg=doi*.

[10] Allen H. Boozer. Control of non-axisymmetric toroidal plasmas. *Plasma Physics and Controlled Fusion*, 52(10):104001, October 2010. ISSN 0741-3335. doi: 10.1088/0741-3335/52/10/104001. URL *http://stacks.iop.org/0741-3335/52/i=10/a=104001?key=crossref.1669f13b7f109f83900d59bf62a2597d*.

[11] C. Cates, Mikhail Shilov, Michael E. Mauel, Gerald A. Navratil, D. Maurer, S. Mukherjee, D. Nadle, J.M. Bialek, and A. Boozer. Suppression of resistive wall instabilities with distributed, independently controlled, active feedback coils. *Physics of Plasmas*, 7(8):3133, 2000. ISSN 1070664X. doi: 10.1063/1.874223. URL *http://link.aip.org/link/doi/10.1063/1.874223/htmlhttp://link.aip.org/link/PHPAEN/v7/i8/p3133/s1&Agg=doi*.

[12] Cory J. Cates. *Active feedback suppression of resistive wall instabilities on HBT-EP*. Phd thesis, Columbia University, 2006.

[13] T. Dudok de Wit, a. L. Pecquet, J.-C. Vallet, and R. Lima. The biorthogonal decomposition as a tool for investigating fluctuations in plasmas. *Physics of Plasmas*, 1(10):3288, 1994. ISSN 1070664X. doi: 10.1063/1.870481. URL *http://link.aip.org/link/PHPAEN/v1/i10/p3288/s1&Agg=doi*.

[14] Shalom Eliezer and Yaffa Eliezer. *The Fourth State of Matter*. IOP Publishing, 2nd editio edition, 2001.

[15] Todd E. Evans, Richard a. Moyer, Keith H. Burrell, Max E. Fenstermacher, Ilon Joseph, Anthony W. Leonard, Thomas H. Osborne, Gary D. Porter, Michael J. Schaffer, Philip B. Snyder, Paul R. Thomas, Jonathan G. Watkins, and William P. West. Edge stability and

transport control with resonant magnetic perturbations in collisionless tokamak plasmas. *Nature Physics*, 2(6):419–423, May 2006. ISSN 1745-2473. doi: 10.1038/nphys312. URL *http://www.nature.com/doifinder/10.1038/nphys312*.

[16] Richard Fitzpatrick. A simple model of the resistive wall mode in tokamaks. *Physics of Plasmas*, 9(8):3459, 2002. ISSN 1070664X. doi: 10.1063/1.1491254. URL *http://link.aip.org/link/PHPAEN/v9/i8/p3459/s1&Agg=doi*.

[17] Richard Fitzpatrick and A. Y. Aydemir. Stabilization of the resistive shell mode in tokamaks. *Nuclear Fusion*, 36(1):11–38, January 1996. ISSN 0029-5515. doi: 10.1088/0029-5515/36/1/I02. URL *http://stacks.iop.org/0029-5515/36/i=1/a=I02?key=crossref.11df511d8534f500d2ec9485434fb151*.

[18] Richard Fitzpatrick and J.M. Bialek. Stability of the resistive wall mode in HBT-EP plasmas. *Physics of Plasmas*, 13(7):072512, 2006. ISSN 1070664X. doi: 10.1063/1.2245542. URL *http://link.aip.org/link/PHPAEN/v13/i7/p072512/s1&Agg=doi*.

[19] Jeffrey Freidberg. *Plasma Physics and Fusion Energy*. Cambridge University Press, 2007. ISBN 0-521-85107-6.

[20] Jeffrey P. Freidberg. *Ideal magnetohydrodynamics*. Plenum Press, 1987. ISBN 0306425122.

[21] Tomonari Furukawa, Benjamin Lavis, and Hugh F Durrant-Whyte. Parallel grid-based recursive Bayesian estimation using GPU for real-time autonomous navigation. In *2010 IEEE International Conference on Robotics and Automation*, pages 316–321. IEEE, May 2010. ISBN 978-1-4244-5038-1. doi: 10.1109/ROBOT.2010.5509396. URL *http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5509396*.

[22] A. M Garofalo, A. D. Turnbull, E.J. Strait, M. E. Austin, J.M. Bialek, M.S. Chu, E.D Fredrickson, R. J. La Haye, Gerald A. Navratil, L. L. Lao, E. a. Lazarus, M. Okabayashi, B. W. Rice, S. a. Sabbagh, J. T. Scoville, T. S. Taylor, and M. L. Walker. Stabilization of the external kink and control of the resistive wall mode in tokamaks. *Physics of Plasmas*, 6(5):1893, 1999. ISSN 1070664X. doi: 10.1063/1.873495. URL *http://link.aip.org/link/PHPAEN/v6/i5/p1893/s1&Agg=doi*.

[23] A.M. Garofalo. *Measurement and Interpretation of Eddy Currents Induced in a Segmented Conducting Wall by MHD Instabilities in a Tokamak*. Phd thesis, Columbia University, 1998.

[24] D. A. Gates. *Stabilization of MHD Instabilities Using a Conducting Wall*. Phd thesis, Columbia University, 1993.

[25] Torkel Glad and Lennart Ljung. *Control Theory, Multivariable and Nonlinear Methods*. Taylor & Francis, New York, 2000. ISBN 0-7484-0877-0.

[26] R.J Goldston and P. H. Rutherford. *Introduction to Plasma Physics*. Institute of Physics Publishing, London, 1995.

[27] Y Gribov, D Humphreys, K Kajiwara, E.a Lazarus, J.B Lister, T Ozeki, A Portone, M Shimada, A.C.C Sips, and J.C Wesley. Progress in ITER Physics Basis: Chapter 8: Plasma operation and control. *Nuclear Fusion*, 47(6):S385–S403, June 2007. ISSN 0029-5515. doi: 10.1088/0029-5515/47/6/S08. URL *http://stacks.iop.org/0029-5515/47/i=6/a=S08?key=crossref.3f3c2344698230634d5ffcfded8ef3b3*.

[28] Michael Griebel and Peter Zaspel. A multi-GPU accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations. *Computer Science - Research and Development*, 25(1-2):65–73, April 2010. ISSN 1865-2034. doi: 10.1007/s00450-010-0111-7. URL *http://www.springerlink.com/index/10.1007/s00450-010-0111-7*.

[29] Eitan Grinspun, Petr Krysl, and P Schroder. CHARMS: a simple framework for adaptive simulation. *ACM Transactions on Graphics*, pages 281–290, 2002. URL *http://docis.info/docis/lib/goti/rclis/dbl/acmgra/(2002)21%253A3%253C281%253ACASFFA%253E/online.cs.nps.navy.mil%252FDistanceEducation%252Fonline.siggraph.org%252F2002%252FPapers%252F02_ModelingAndSimulation%252Fpapers%252Fgrinspun.pdf*.

[30] R. A. Gross. *Fusion Energy*. Wiley, 1984. ISBN 0-471-88470-7.

[31] Jeremy M. Hanson. *A Kalman Filter for Active Feedback on Rotating External Kink Instabilities in a Tokamak Plasma*. Phd thesis, Columbia University, 2009. URL *http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:A+Kalman+Filter+for+Active+Feedback+on+Rotating+External+Kink+Instabilities+in+a+Tokamak+Plasma#0*.

[32] Jeremy M. Hanson, Bryan De Bono, Royce W. James, Jeffrey P. Levesque, Michael E. Mauel, David A. Maurer, Gerald A. Navratil, Thomas Sunn Pedersen, and Daisuke Shiraki. Feedback suppression of rotating external kink instabilities in the presence of noise. *Physics of Plasmas*, 15(8):080704, August 2008. ISSN 1070664X. doi: 10.1063/1.2974797.

URL *http://link.aip.org/link/doi/10.1063/1.2974797/htmlhttp://link.aip.org/link/?PHPAEN/15/080704/1*.

[33] Jeremy M. Hanson, Bryan De Bono, Jeffrey P. Levesque, Michael E. Mauel, David A. Maurer, Gerald A. Navratil, Thomas Sunn Pedersen, Daisuke Shiraki, and Royce W. James. A Kalman filter for feedback control of rotating external kink instabilities in the presence of noise. *Physics of Plasmas*, 16(5):056112, 2009. ISSN 1070664X. doi: 10.1063/1.3110110. URL *http://link.aip.org/link/PHPAEN/v16/i5/p056112/s1&Agg=doi*.

[34] Jeremy M. Hanson, Alexander J. Klein, Michael E. Mauel, David A. Maurer, Gerald A. Navratil, and Thomas Sunn Pedersen. A digital control system for external magnetohydrodynamic modes in tokamak plasmas. *The Review of scientific instruments*, 80(4):043503, April 2009. ISSN 1089-7623. doi: 10.1063/1.3112607. URL *http://www.ncbi.nlm.nih.gov/pubmed/19405656*.

[35] K Ikeda. Progress in the ITER Physics Basis. *Nuclear Fusion*, 47(6), June 2007. ISSN 0029-5515. doi: 10.1088/0029-5515/47/6/E01. URL *http://stacks.iop.org/0029-5515/47/i=6/a=E01?key=crossref.3391a8dabcfa415ae345d3defa8b1c9f*.

[36] Y In, M.S. Chu, G L Jackson, J S Kim, R J La Haye, Y Q Liu, L Marrelli, M Okabayashi, H Reimerdes, and E.J. Strait. Requirements for active resistive wall mode (RWM) feedback control. *Plasma Physics and Controlled Fusion*, 52(10):104004, October 2010. ISSN 0741-3335. doi: 10.1088/0741-3335/52/10/104004. URL *http://stacks.iop.org/0741-3335/52/i=10/a=104004*.

[37] O. Katsuro-Hopkins, J. Bialek, D.a. Maurer, and G.a. Navratil. Enhanced ITER resistive wall mode feedback performance using optimal control techniques. *Nuclear Fusion*, 47 (9):1157–1165, September 2007. ISSN 0029-5515. doi: 10.1088/0029-5515/47/9/012. URL *http://stacks.iop.org/0029-5515/47/i=9/a=012?key=crossref.08ec788bab3a083c0fe571b6f136199f*.

[38] Alexander J. Klein, David A. Maurer, Thomas Sunn Pedersen, Michael E. Mauel, Gerald A. Navratil, Cory Cates, Mikhail Shilov, Yuhong Liu, Nikolai Stillits, and J.M. Bialek. Suppression of rotating external kink instabilities using optimized mode control feedback. *Physics of Plasmas*, 12(4):040703, 2005. ISSN 1070664X. doi: 10.1063/1.1868732. URL *http://link.aip.org/link/PHPAEN/v12/i4/p040703/s1&Agg=doi*.

[39] SE Kruger and CC Hegna. Reduced MHD Equations for Low Aspect Ratio Plasmas. In *Proceedings of the 1998 International Congress on Plasma Physics & 25th EPS Conference on Controlled Fusion and Plasma Physics*, volume 22, pages 2034–2037, 1998. URL *http://epsppd.epfl.ch/Praha/WEB/98ICPP_W/Go96PR.PDF*.

[40] Jeffrey P Levesque. *Multimode Structure of Resistive Wall Modes near the Ideal Wall Stability Limit*. Phd thesis, Columbia University, 2012.

[41] Y. Q. Liu, a. Bondeson, C. M. Fransson, B. Lennartson, and C. Breitholtz. Feedback stabilization of nonaxisymmetric resistive wall modes in tokamaks. I. Electromagnetic model. *Physics of Plasmas*, 7(9):3681, 2000. ISSN 1070664X. doi: 10.1063/1.1287744. URL *http://link.aip.org/link/PHPAEN/v7/i9/p3681/s1&Agg=doi*.

[42] Michael E. Mauel. Discussion of the RWM Eigensystem, 2004.

[43] Michael E. Mauel, J.M. Bialek, Allen H. Boozer, C. Cates, R. James, Oksana Katsuro-Hopkins, Alexander J. Klein, Y. Liu, David A. Maurer, D. Maslovsky, Gerald A. Navratil, Thomas Sunn Pedersen, Mikhail Shilov, and N. Stillits. Dynamics and control of resistive wall modes with magnetic feedback control coils: experiment and theory. *Nuclear Fusion*, 45(4):285–293, April 2005. ISSN 0029-5515. doi: 10.1088/0029-5515/45/4/010. URL *http://stacks.iop.org/0029-5515/45/i=4/a=010?key=crossref.d2377b08b0b5c069272b9c6dbffe0b13*.

[44] D. A. Maurer, J.M. Bialek, P. J. Byrne, B. Debono, J. P. Levesque, B. Q. Li, M. E. Mauel, G. A. Navratil, T. S. Pedersen, N. Rath, and D. Shiraki. The high beta tokamak-extended pulse magnetohydrodynamic mode control research program. *Plasma Physics and Controlled Fusion*, 53(7):074016, July 2011. ISSN 0741-3335. doi: 10.1088/0741-3335/53/7/074016. URL *http://stacks.iop.org/0741-3335/53/i=7/a=074016?key=crossref.doc5612118b08871a72c2329184128a4*.

[45] Dirk Naujoks. *Plasma-Material Interaction in Controlled Fusion*. Springer, 2006. ISBN 3-540-32148-9.

[46] K. Nishikawa and M. Wakatani. *Plasma Physics*. Springer, 3rd edition, 2000. ISBN 3-540-65285-X.

[47] NVidia. NVIDIA CUDA C Programming Guide, 2012. URL *http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf*.

[48] M. Okabayashi, I.N. Bogatu, M.S. Chance, M.S. Chu, A.M. Garofalo, Y. In, G.L. Jackson, R.J. La Haye, M.J. Lanctot, J. Manickam, L. Marrelli, P. Martin, Gerald A. Navratil, H. Reimerdes, E.J. Strait, H. Takahashi, A.S. Welander, T. Bolzonella, R.V. Budny, J.S. Kim, R. Hatcher, Y.Q. Liu, and T.C. Luce. Comprehensive control of resistive wall modes in DIII-D advanced tokamak plasmas. *Nuclear Fusion*, 49(12):125003, December 2009. ISSN 0029-5515. doi: 10.1088/0029-5515/49/12/125003. URL *http://stacks.iop.org/0029-5515/49/i=12/a=125003?key=crossref.16e6a062fe1c5a11cb7c4b1585f545f2*.

[49] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Timothy J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, 26(1):80–113, March 2007. ISSN 0167-7055. doi: 10.1111/j.1467-8659.2007.01012.x. URL *http://doi.wiley.com/10.1111/j.1467-8659.2007.01012.x*.

[50] J. Park, Allen H. Boozer, and A.H. Glasser. Computation of three-dimensional tokamak and spherical torus equilibria. *Physics of Plasmas*, 14:052110, 2007. doi: 10.1063/1.2732170. URL *http://link.aip.org/link/?php/14/052110*.

[51] Jong-kyu Park. *Ideal Perturbed Equilibria in Tokamaks*. PhD thesis, Princeton University, 2009.

[52] N. Rath, J. Bialek, P.J. Byrne, B. DeBono, J.P. Levesque, B. Li, M.E. Mauel, D.A. Maurer, G.A. Navratil, and D. Shiraki. High-speed, multi-input, multi-output control using GPU processing in the HBT-EP tokamak. *Fusion Engineering and Design*, null(null), May 2012. ISSN 09203796. doi: 10.1016/j.fusengdes.2012.04.003. URL *http://dx.doi.org/10.1016/j.fusengdes.2012.04.003*.

[53] Guochun Shi, Steven Gottlieb, Aaron Torok, and Volodymyr Kindratenko. Design of MILC Lattice QCD Application for GPU Clusters. In *2011 IEEE International Parallel & Distributed Processing Symposium*, pages 363–371. IEEE, May 2011. ISBN 978-1-61284-372-8. doi: 10.1109/IPDPS.2011.43. URL *http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6012807*.

[54] M. Shilov, C. Cates, R. James, a. Klein, O. Katsuro-Hopkins, Y. Liu, M. E. Mauel, D. a. Maurer, G. a. Navratil, T. S. Pedersen, N. Stillits, R. Fitzpatrick, and S. F. Paul. Dynamical plasma response of resistive wall modes to changing external magnetic perturbations. *Physics of Plasmas*, 11(5):2573, 2004. ISSN 1070664X. doi: 10.1063/1.1688793. URL *http://link.aip.org/link/PHPAEN/v11/i5/p2573/s1&Agg=doi*.

[55] Mikhail Shilov. *Measurement of the Response of External Kink Modes to Resonant Magnetic Perturbation in HBT-EP Tokamak Plasmas*. Phd thesis, Columbia University, 2005.

[56] Daisuke Shiraki. *High-Resolution MHD Spectroscopy of External Kinks in a Tokamak Plasma*. Phd thesis, Columbia University, 2012.

[57] H. R. Strauss. Finite-aspect-ratio MHD Equations for Tokamaks. *Nuclear Fusion*, 23:649, 1983.

[58] Z. Sun, a. K. Sen, and R. W. Longman. Adaptive stochastic output feedback control of resistive wall modes in tokamaks. *Physics of Plasmas*, 13(9):092508, 2006. ISSN 1070664X. doi: 10.1063/1.2355451. URL *http://link.aip.org/link/PHPAEN/v13/i1/p012512/s1&Agg=doihttp://link.aip.org/link/PHPAEN/v13/i9/p092508/s1&Agg=doi*.

[59] Natalya Tatarchuk, Jeremy Shopf, and Christopher DeCoro. Advanced interactive medical visualization on the GPU. *Journal of Parallel and Distributed Computing*, 68(10):1319–1328, October 2008. ISSN 07437315. doi: 10.1016/j.jpdc.2008.06.011. URL *http://linkinghub.elsevier.com/retrieve/pii/S0743731508001226*.

[60] John Wesson. *Tokamaks*. Oxford University Press, 2011. ISBN 0199592233.

[61] Robert L. Williams II and Douglas A. Lawrence. *Linear State Space Control Systems*. Wiley, New Jersey, 2007. ISBN 978-0-471-73555-7.